Voronoi图及其应用

杨承磊　吕琳　杨义军　孟祥旭　著

清华大学出版社

Voronoi 图及其应用

杨承磊　吕琳
杨义军　孟祥旭　著

清华大学出版社

北京

内 容 简 介

本书在介绍 Voronoi 图相关概念和性质的基础上，侧重介绍 Voronoi 图的构造和应用方面的算法。本书主要内容包括离散点集的 Voronoi 图与 Delaunay 三角部分、多边形的 Voronoi 图、约束 Delaunay 三角部分以及重心 Voronoi 图的基本概念、性质、构造算法，及其在多边形剖分、几何搜索、多边形求交、可见性计算、路径规划、碰撞检测、骨架计算、文字特征提取、半色调图像生成以及信息可视化等方面的应用。

本书可以供从事相关研究的高校教师、科研人员参考，也可作为高等院校计算机相关专业研究生的教材和参考书。

图书在版编目（CIP）数据

Voronoi 图及其应用/杨承磊等著. --北京：清华大学出版社，2013
ISBN 978-7-302-32993-0

Ⅰ. ①V…　Ⅱ. ①杨…　Ⅲ. ①空间测量　Ⅳ. ①P236

中国版本图书馆 CIP 数据核字（2013）第 148236 号

责任编辑： 付弘宇
封面设计： 傅瑞学
责任校对： 焦丽丽
责任印制： 宋　林

出版发行： 清华大学出版社
网　　址： http://www.tup.com.cn，　http://www.wqbook.com
地　　址： 北京清华大学学研大厦 A 座　　**邮　　编：** 100084
社 总 机： 010-62770175　　**邮　　购：** 010-62786544
投稿与读者服务： 010-62776969, c-service@tup.tsinghua.edu.cn
质量反馈： 010-62772015, zhiliang@tup.tsinghua.edu.cn
课件下载： http://www.tup.com.cn,010-62795954
印 刷 者： 北京鑫丰华彩印有限公司
装 订 者： 三河市新茂装订有限公司
经　　销： 全国新华书店
开　　本： 160mm×230mm　　**印　张：** 11　　**字　　数：** 167 千字
版　　次： 2013 年 10 月第 1 版　　**印　　次：** 2013 年 10 月第 1 次印刷
印　　数： 1～500
定　　价： 30.00 元

产品编号： 053611-01

作 者 简 介



杨承磊，男，于 1995、1998、2004 年先后获得山东大学计算机应用专业理学学士学位、计算机软件与理论专业工学硕士和博士学位。目前为山东大学计算机科学与技术学院教授。2007 年 1～7 月在香港大学计算机科学系开展合作研究，2010 年 6 月至 2011 年 6 月在哈佛大学做访问学者。

研究主要围绕工业 CAD、文化与自然遗产保护、数字娱乐与远程教育等应用领域，重点开展离散计算几何、人机交互与虚拟现实等方面的理论研究与项目研发工作。先后主持国家自然科学基金 3 项、国家支撑计划课题 1 项和省部级项目 4 项，并作为学术骨干参与完成了国家 973 计划、863 计划等 10 多项国家级、省部级科研课题，作为骨干成员参与研发出“集成化计算机辅助图案设计与制版系统”等系统软件，获得国家科技进步奖二等奖 1 项、教育部科技进步奖二等奖 1 项以及山东省科技进步奖三等奖 1 项。目前在 *CAD*、*C&G*、*The Visual Computer*、中国科学、软件学报、计算机学报等国内外主要学术刊物与会议上发表论文 50 余篇。与他人合著《Voronoi 图及其应用》、《计算几何及应用》、《人机交互基础教程（第 2 版）（普通高等教育“十一五”国家级规划教材）》等专著和教材。

前　言

Voronoi 图是一种重要的几何结构，也是计算几何领域重要的研究内容之一。Voronoi 图既是一种行之有效的空间剖分和聚类方法，又具有骨架的特性。它按照站点（sites）集合中元素的最近邻属性将空间划分成许多单元区域。在不同应用背景下，根据生成空间、测量距离以及站点等定义条件的不同，又产生了不同类型的 Voronoi 图。这些都使得 Voronoi 图在机器人、CAD、模式识别、虚拟现实、地理信息系统、数据挖掘、生物计算以及无线传感网络等领域得到广泛应用，也是解决碰撞检测、路径规划、可见性计算、骨架计算以及凸包计算等计算几何领域中其他问题的有效工具。

本书在介绍 Voronoi 图的概念、性质以及一些经典构造算法的基础上，侧重介绍我们近十年的相关研究成果。本书主要内容包括离散点集的 Voronoi 图与 Delaunay 三角剖分、多边形的 Voronoi 图、约束 Delaunay 三角剖分以及重心 Voronoi 图的基本概念、性质、构造算法，及其在几何搜索、多边形求交、可见性计算、路径规划、碰撞检测、骨架计算、字符特征提取、骨架匹配与模型分割、半色调图像生成以及信息可视化等方面的应用。

本书在注重介绍 Voronoi 图相关基础理论的同时，尽量提供一些简洁、实用和易编程的算法，目的是力求让读者易读、易懂，在学习了本书以后，能解决在各种应用领域中遇到的相关问题。

本书成果与写作过程得到了国家自然科学基金委项目（69873028，60703028，61070093，61272243，61202147）的持续资助。我们在此表示衷心的感谢。

本书由杨承磊、吕琳、杨义军以及孟祥旭合著而成，杨承磊、孟祥旭统稿和审定。由于时间仓促，编者水平有限，书中欠妥和纰漏之处在所难免，恳请读者和同行不吝指正。

作者

2013 年 5 月

目　录

第 1 章　引　论

本章主要对Voronoi图（Voronoi Diagram）的起源、类型及其应用等进行概述，并介绍后面章节涉及到的一些基本概念和知识。

1.1　Voronoi 图概述

计算几何（Computational Geometry）作为一门学科，起源于20世纪70年代，经过近四十多年的发展，其研究内容不断扩大，涉及Voronoi图、三角剖分、凸包、直线与多边形求交、可见性、路径规划、多边形剖分等内容。据相关统计，在数以千计的相关文章中，约有15%是关于Voronoi图及其对偶图——Delaunay三角剖分（Delaunay Triangulation）的研究。由于Voronoi图具有最近性、邻接性等众多性质和比较系统的理论体系，如今已经在计算机图形学、机械工程、地理信息系统、机器人、图像处理、大数据分析与处理、生物计算以及无线传感网络等领域得到了广泛应用，同时也是解决碰撞检测、路径规划、可见性计算、骨架计算以及凸包计算等计算几何所涉及的其他问题的有效工具[WangJY2011，Okabe2000，Gold2006，Chen2004，Fan2010，Held1991，Aurenhammer1991等]。

Voronoi图的起源最早可以追溯到17世纪。1644年，Descartes用类似Voronoi图的结构显示太阳系中物质的分布[Descartes1644]。数学家G. L. Dirichlet [Dirichlet1850]和M.G.Voronoi[Voronoi1908]分别于1850年和1908年在他们的论文中讨论了Voronoi图的概念，所以Voronoi图又叫Dirichlet tessellation。在其他领域，这个概念也曾分别独立地出现，如生物学和生理

学中称之为中轴变换（Medial Axis Transform）或骨架（Skeleton），化学与物理学中称之为Wigner-Seitz Zones，气象学和地理学中称之为Thiessen多边形[Thiessen1911]（Voronoi图最早由Thiessen应用于气象观测站中随机分布的研究）等。由于M.G.Voronoi从更通用的n维情况对其进行研究和定义，所以Voronoi图这个名称为大多数人所使用。

Voronoi图是计算几何中一种仅次于凸包的重要几何结构，也是计算几何的重要研究内容之一。它按照站点（Site）集合中元素的最近邻属性将空间划分成许多单元区域，即Voronoi区域。Voronoi区域与其生成站点一一对应且互不重叠，构成空间的一个划分。根据站点的Voronoi区域是否邻接，就可以定义站点间的邻接关系。

早期的研究工作更多地集中在离散点集的Voronoi图。n个平面上的离散的点，按照最近邻原则划分平面，每个点与它的最近邻区域相关联。Delaunay三角形是由与相邻Voronoi多边形共享一条边的相关点连接而成的三角形。Delaunay三角形的外接圆圆心是与三角形相关的Voronoi多边形的一个顶点。Voronoi图是Delaunay三角形的对偶图。

在不同应用背景下，根据生成空间、测量距离以及生成站点等条件的不同，Voronoi图有着各种各样的定义[WangJY2011，Okabe2000]。例如，根据空间可分成二维、三维以及更高维Voronoi图。最远点Voronoi图则是由到一个离散点距离最远的点构成的Voronoi区域（如图1.1所示）。又如，同时到离散点集中的两个点的距离最小的点构成的区域，称之为二阶Voronoi图（如图1.2所示）。或者更一般地，定义一个k阶Voronoi图：对于一个有n个点的点集，给定整数$k(0<k<n)$，同时到点集中k个点距离最小的点所划分的区域构成了k阶Voronoi图。另外，通过给每个站点赋予不同的权值，可得到加权的Voronoi图（如图1.3所示），Power图是其中的一种。在路径规划、机械加工、模式识别、虚拟现实、生物计算等领域，将站点从离散点扩展到线段、圆弧等生成Voronoi图的方式也是非常常见的（如图1.4所示）。目前也有不少

研究工作是关于PSLG（Planar Straight Line Graph，平面直线图）的Voronoi图及其应用。PSLG的Voronoi图有不同的称谓，如线状Voronoi图、多边形的中轴或骨架等。因此，有人认为多边形的Voronoi图起源于1967年Blum关于中轴（骨架）的工作（Blum首先用中轴表示连续平面图形）[Blum1967]。

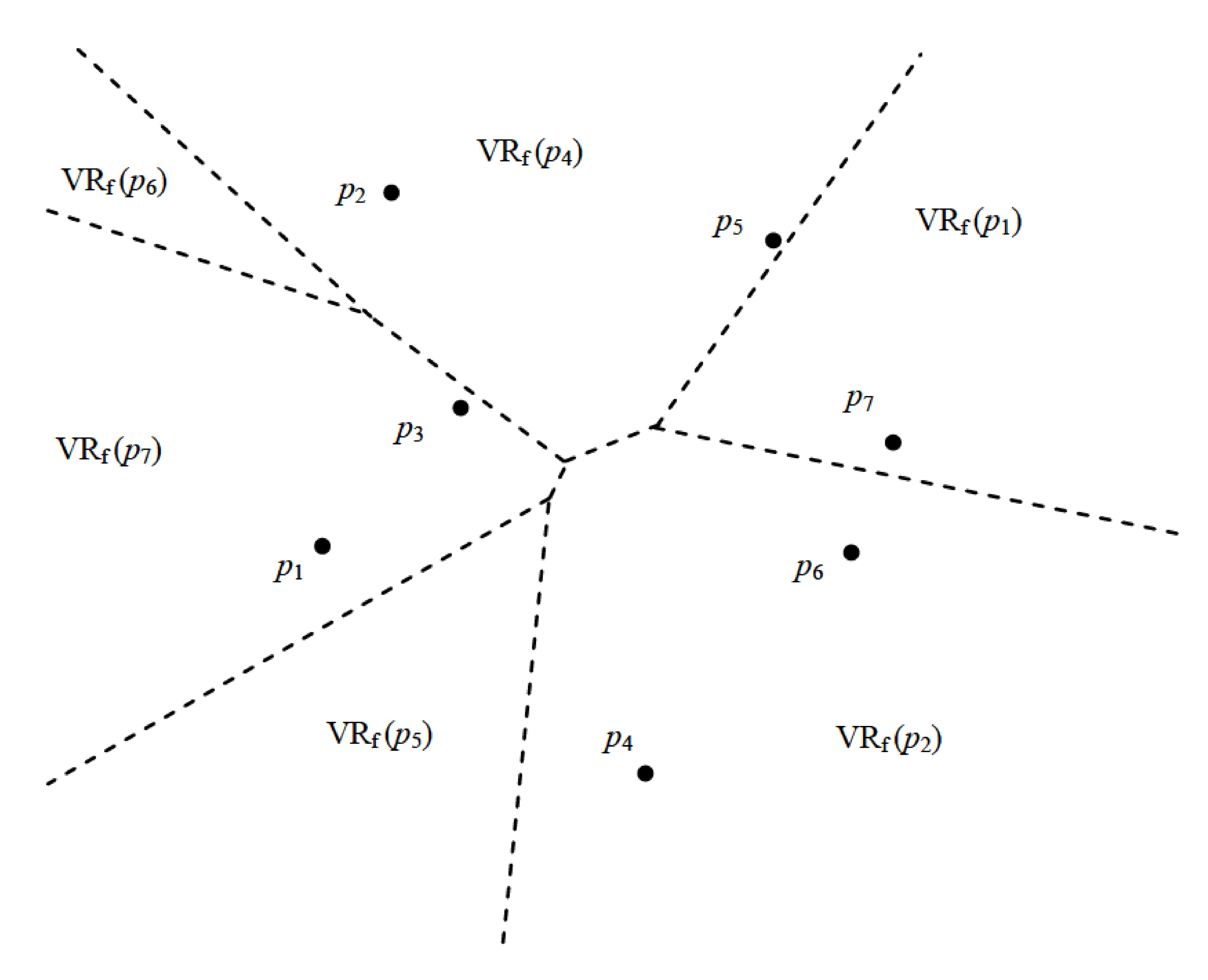


图1.1　最远点Voronoi图，其中$VR_f(p_i)$表示p_i的Voronoi区域

目前，有关Voronoi图的专门的国际会议为每年举行一次的ISVD会议（International Symposium on Voronoi Diagrams in Science and Engineering），它已有十年的历史。有关Voronoi图的专著主要包括文献[Wang JY2011，Okabe2000，Cai2010，Edelsbrunner2001，Hjelle2006，Sack2000，O'Rourke1994，Preparata1991，Berg2000，Goodman2004，YangQ2005，Zhou2008，Li2010]等。计算几何算法库（Computational Geometry Algorithms Library，CGAL）提供了一些关于Voronoi图和Delaunay三角剖分的算法软件[CGAL]。

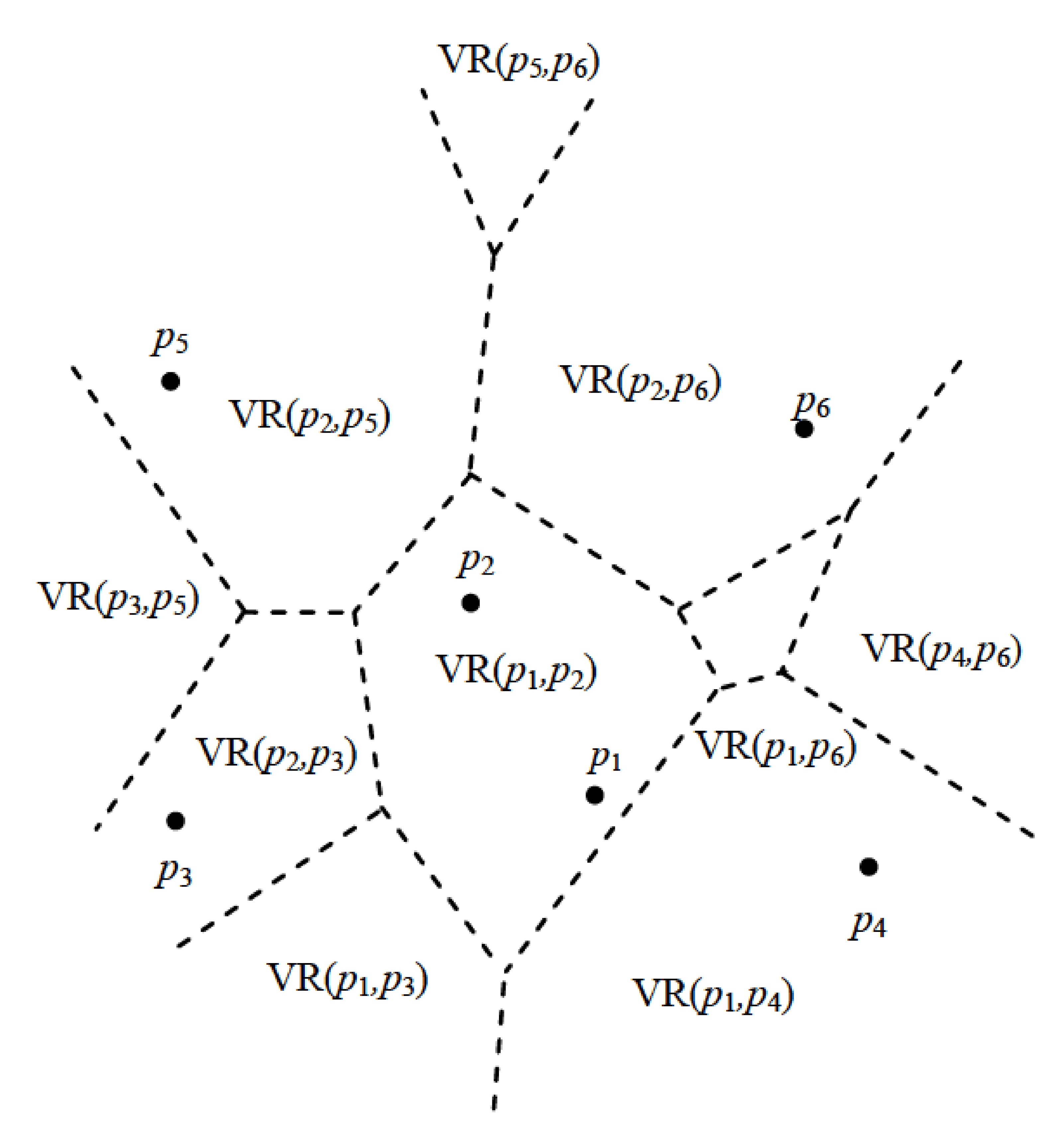


图 1.2　二阶 Voronoi 图，其中 p_i 表示给定的点，VR(p_i,p_j) 表示站点(p_i,p_j)的 Voronoi 区域

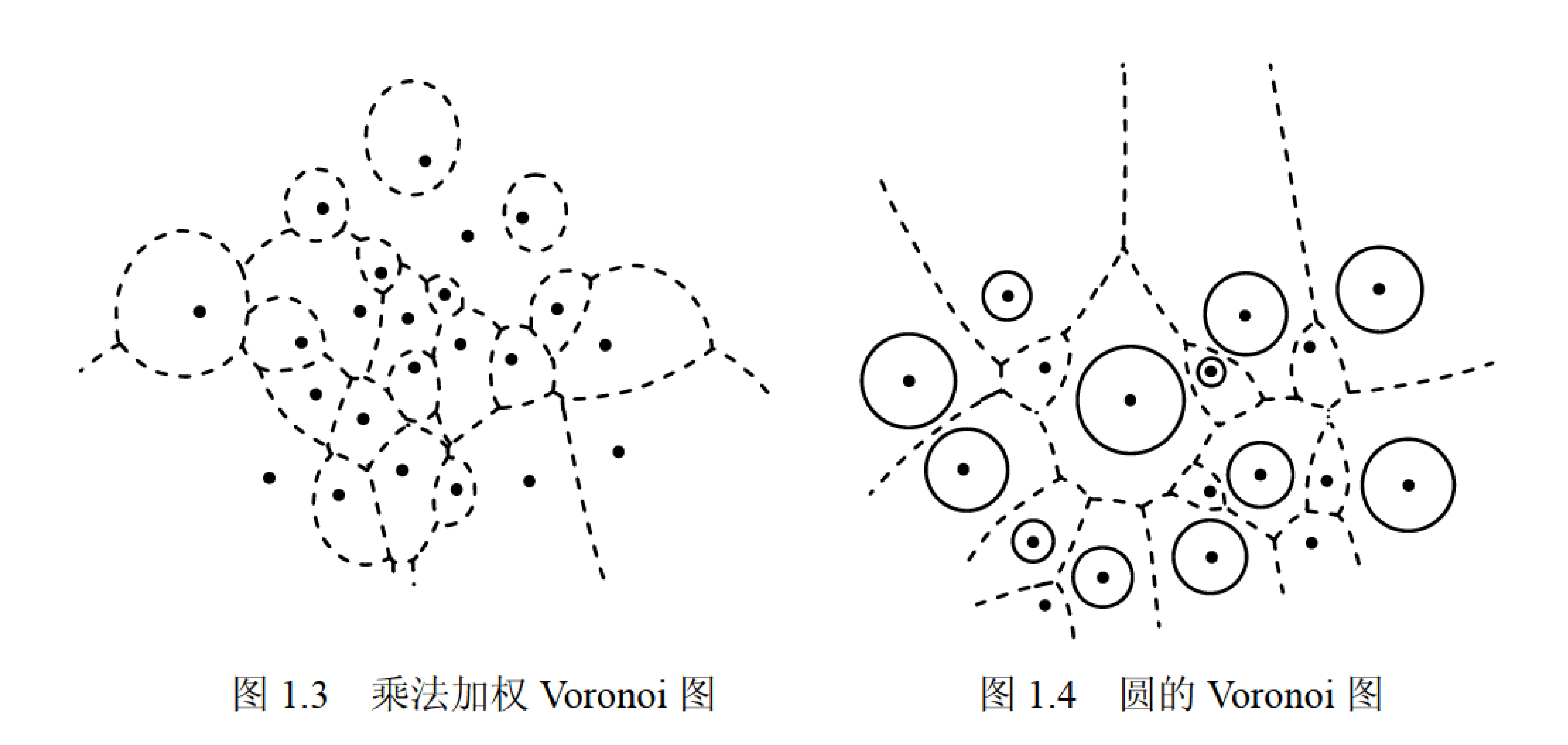


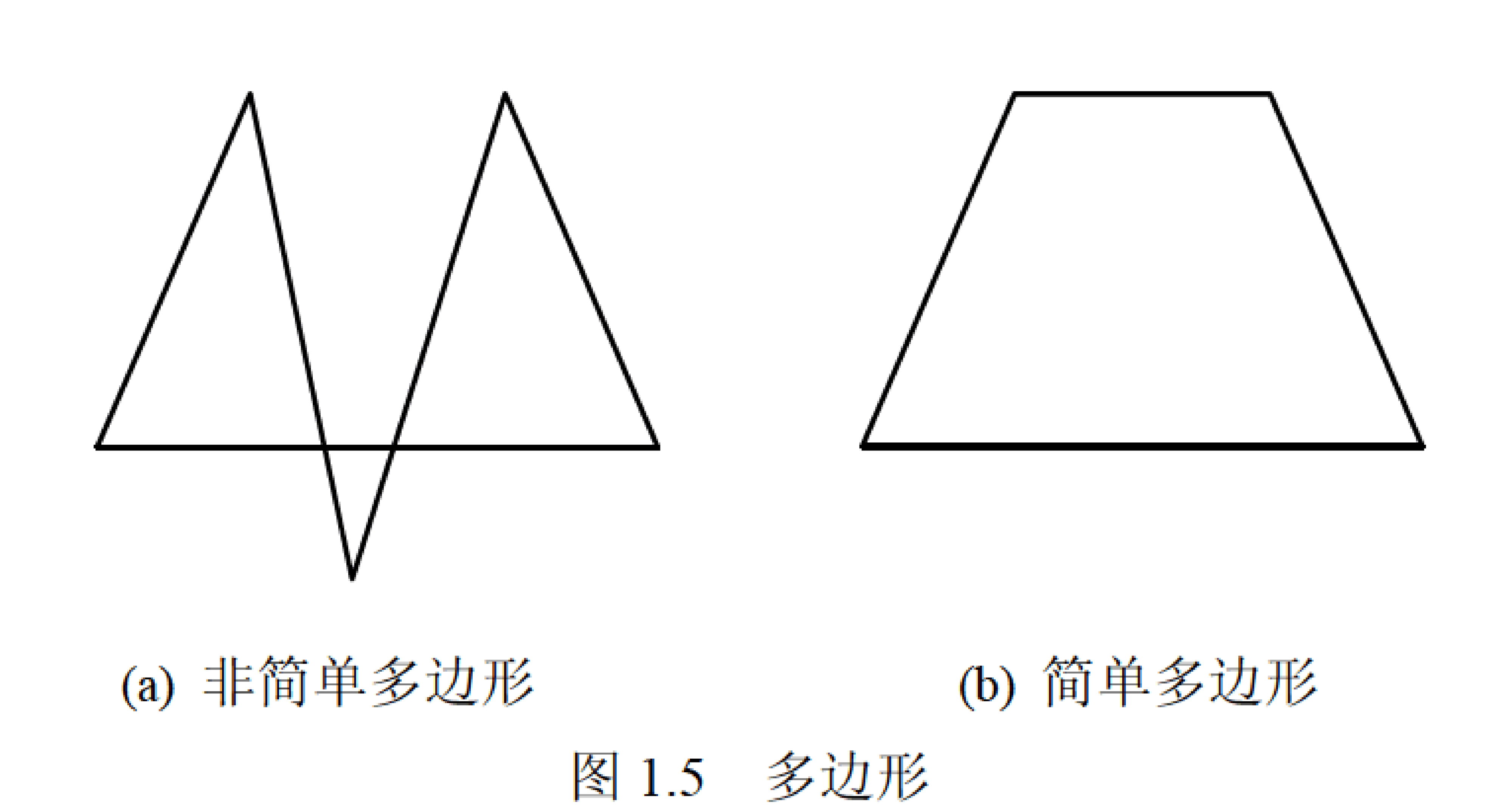
图 1.3　乘法加权 Voronoi 图　　图 1.4　圆的 Voronoi 图

1.2　相关基础概念

定义1.1　在二维欧氏空间 E^2 中的一个多边形是用一个有限线段的集合 L 来定义的，L 中每一线段的每一个端点是 L 中两条线段且仅是这两条线段

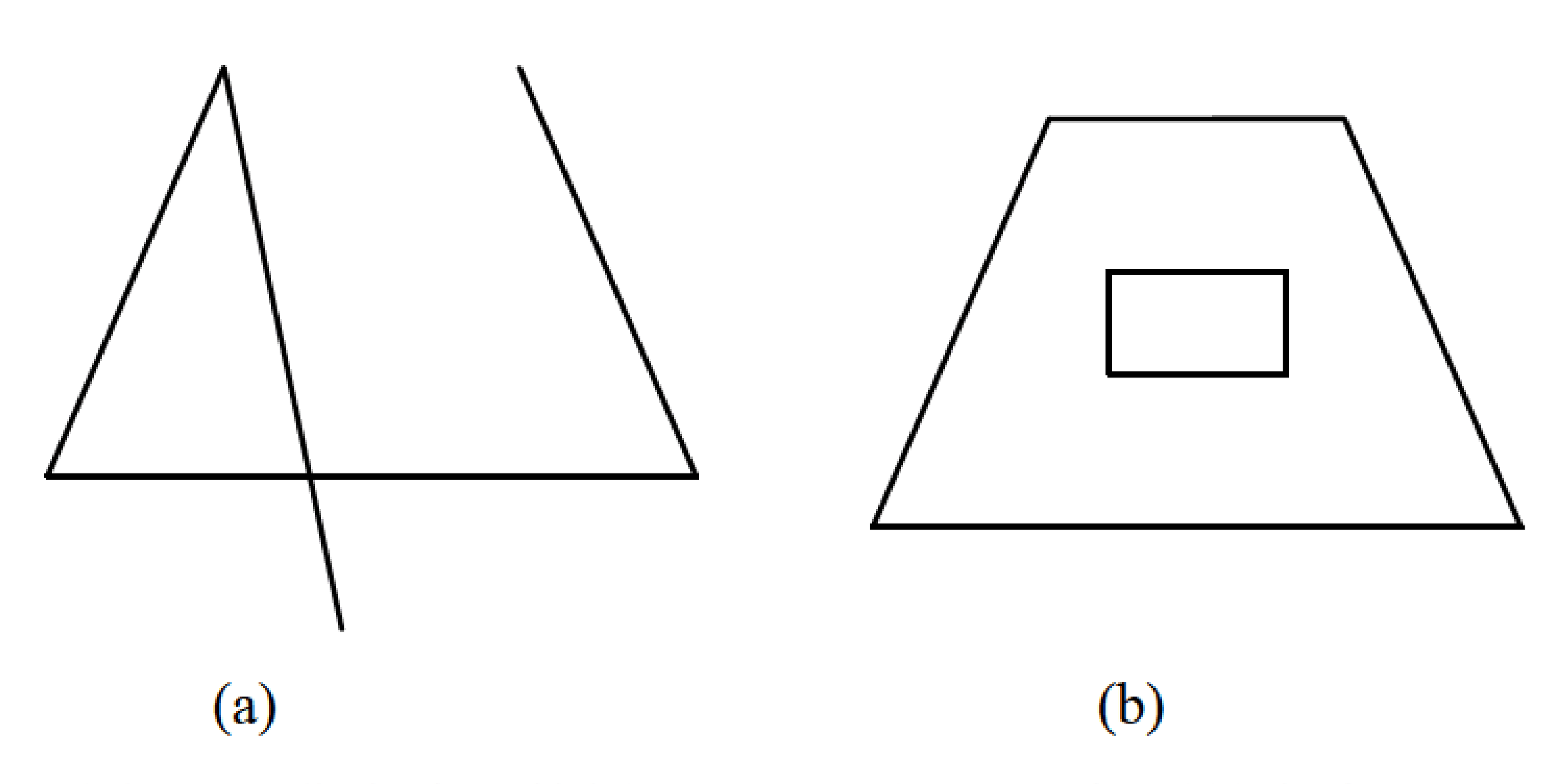
的公共端点，L 中任两条线段上的点均可通过集合 L 中的线段连通。构成多边形的线段称为多边形的边，边的端点称为多边形的顶点。

图1.5中的线段集合构成多边形，但图1.6中的线段集合不构成多边形。由多边形定义可知，多边形是线段集合的一个有序的排列。有公共端点的两条边被称为相邻边。在本书的后面章节会讨论类似图1.6（b）中图形所围成的区域，为了区分，我们将这样的图形称为多边界多边形。



(a) 非简单多边形　　(b) 简单多边形

图 1.5　多边形



(a)　　(b)

图 1.6　非多边形的例子，（a）和（b）都不是多边形

定义 1.2　如果一个多边形的任何不相邻的两条边均不相交，则称这个多边形为简单多边形。一个简单多边形只有一条边界，我们也称之为单边界多边形。图 1.5（a）是多边形但不是简单多边形，图 1.5（b）是简单多边形。

定义 1.3　简单多边形把平面分割成内部和外部两个不连通的区域。如果一个简单多边形的内部是凸集，则这个简单多边形是凸的，称为凸多边形。

定义 1.4 我们将由多条简单的、封闭的、不相交的曲线围成的平面区域称为多边界多边形。这些曲线都是它的边界。一个多边界多边形有多个边界。

一般地，多边界多边形有下面两种类型。

（1）第一类多边界多边形是一些分离的单边界多边形的集合被另一个单边界多边形所包围，这一个最外面的边界称之为外边界，其他的单边界多边形都在它的内部，我们称之为内边界。根据单边界多边形的定义，这些多边形内部不存在任何多边形了。内边界和外边界之间的部分是我们感兴趣的部分，称为多边形的关注区域。

这类多边界多边形在机械加工中被称为 Pocket[Held1991]，在数学上被称为多连通多边形。

（2）第二类多边界多边形可以想象成第一类多边界多边形的外边界被不断放大并移向无穷，在平面上留下若干个分离的单边界多边形的集合。这些多边形的外部并延伸至无限的部分则是我们感兴趣的部分，为多边形的关注区域。

对于第一类多边界多边形，规定外边界顶点是以逆时针方向排序，内边界顶点是以顺时针方向排序。对第二类多边界多边形，边界顶点都是以顺时针方向排序。如果一个人沿这些边界走动，多边形关注区域总是位于前进方向的左侧。

为了便于叙述，本书中多边界多边形简称为多边形，或称为复杂多边形。

定义 1.5 对于多边形的一条边界上的一个顶点，令它的两个关联边的夹角为 α。该夹角的大小是计算扫过多边形的关注区域的角度，如果 $\alpha > 180°$，则称该顶点为（向外）凹顶点，也称为内尖点；如果 $\alpha < 180°$，则称该顶点为（向外）凸顶点；如果 $\alpha = 180°$，则称该顶点为切顶点。

定义 1.6 在一个简单多边形 P 中，如果存在这样一个点 z，对 P 内任意一点 p，都使线段 zp 完全位于 P 内，则称 P 为星形多边形。

定义 1.7　图论中的图 $G=(P, E)$是由顶点集合 P 和边集合 E 组成的。若能把 P 的所有顶点映射到一个平面上，对 E 中的每条边的端点在该平面上的两个映像点，都有平面上的一条对应的简单曲线连接，如果除了在顶点处外，这些简单曲线之间不再有交点，则这个图 $G=(P, E)$称为可嵌到平面上。可嵌到平面上的图称为平面图。边为直线的平面图称为平面直线图（PSLG）。

第 2 章　离散点集的 Voronoi 图及其应用

2.1　定义与性质

2.1.1　定义

对于平面上任意 n 个位置互异的离散点的集合 $P=\{p_1，p_2，\cdots，p_n\}$，其 Voronoi 图存在几种等价的解释与定义，包括最近距离聚类法、通过 Delaunay 三角剖分计算对偶图法、半平面法、波传播法、圆锥投影法、凸包投影法等。

1. 最近距离聚类法

该方法将平面上的所有点按照它们到所给离散点 $p_1，p_2，\cdots，p_n$ 的欧氏距离的远近进行聚类，分别得到每个离散点 $p_i(1\leqslant i\leqslant n)$的最近点的集合：

$$\mathrm{VR}(p_i)=\{q \mid |qp_i|\leqslant|qp_j|，\forall j\neq i，p_i，p_j\in P\}$$

其中，$|qp_i|$表示 q 和 p_i 两点间的欧氏距离[WangJY2011]。

P 的 Voronoi 图为：

$$\mathrm{VD}(P)=\bigcup_{i=1}^{n}\mathrm{VR}(P_i)$$

这里，为了避免与平面上任意一点混淆，我们称 $p_i(1\leqslant i\leqslant n)$为站点，并称 $\mathrm{VR}(p_i)$为 p_i 的 Voronoi 区域（或 Voronoi 多边形、Voronoi 单元）。将 Voronoi 区域的边称之为 Voronoi 边，Voronoi 区域的顶点称之为 Voronoi 顶点，每个

Voronoi 顶点 v 所连接的边数称为 v 的度。每条 Voronoi 边 e 被且只被 2 个站点的 Voronoi 区域所共享，我们将这 2 个站点称之为 e 的关联站点。Voronoi 顶点的度一般为 3，当度大于 3 时便认为是退化情况（为了叙述简洁，若不做特别说明，本书后面部分不再涉及退化情况），这些共享同一个 Voronoi 顶点 v 的 Voronoi 区域的站点被称为 v 的关联站点。图 2.1 给出了 11 个离散点（黑色实点）的 Voronoi 图，其中实线边为 Voronoi 边。

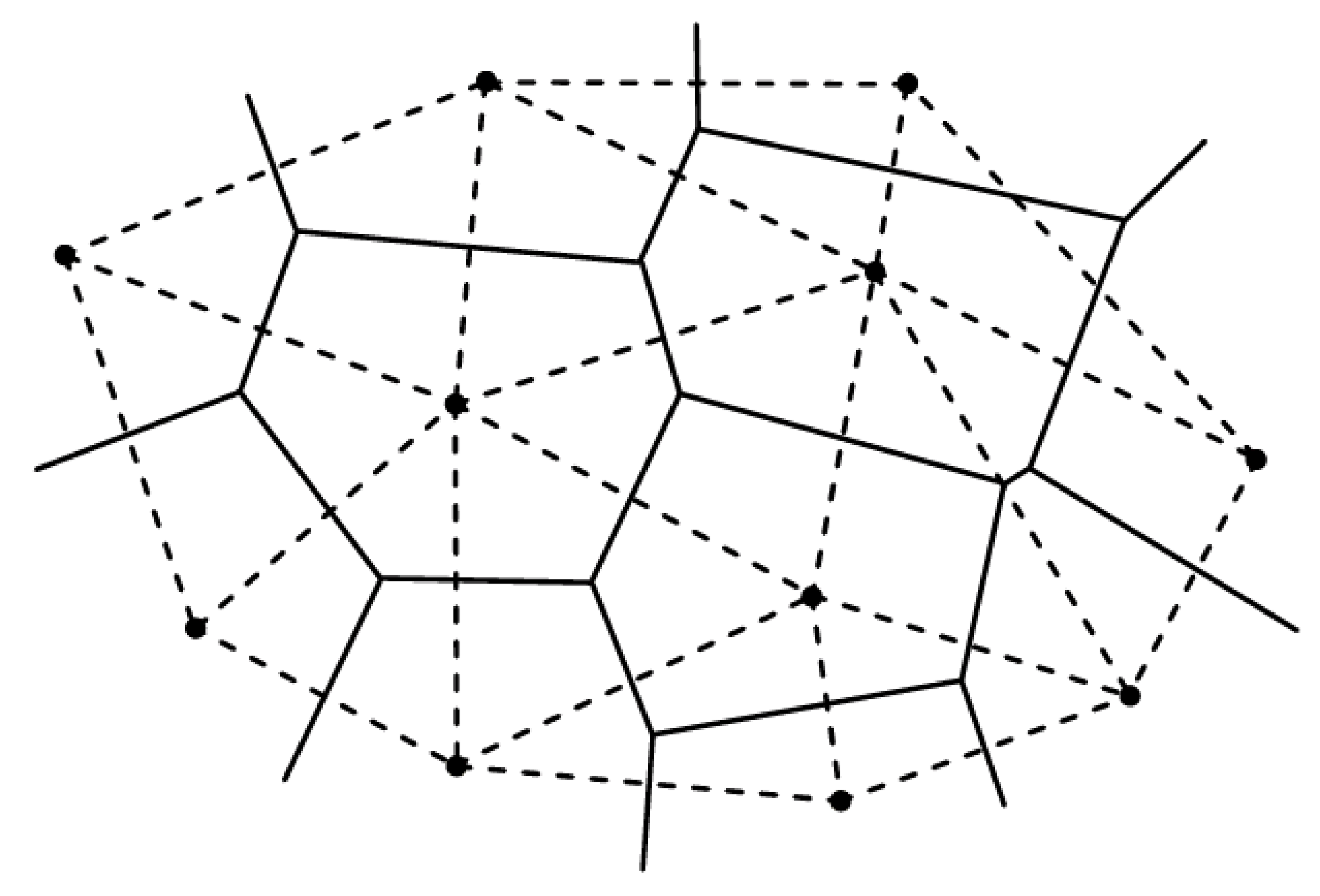


图 2.1　平面离散点集（黑色实点）的 Voronoi 图（实线部分）
及其 Delaunay 三角剖分（虚线部分）

对于一个给定的点集，设其中的若干点为站点，可根据类似前面的 Voronoi 区域的定义计算出到每个站点距离最近的点，从而实现该点集的聚类。

2. 对偶图—Delaunay 三角剖分法

对任一个平面点集 $P=\{p_1, p_2, \cdots, p_n\}$，它的 Voronoi 图用 VD (P)表示，我们将每条 Voronoi 边的两个关联站点(即两个站点的 Voronoi 区域有共享边)都用直线段（如图 2.1 中虚线所示）连接在一起，就可得到 P 的 Voronoi 图的对偶图——Delaunay 图。在退化情况下，即存在四点共圆时，Delaunay 图中存在不是三角形的面；在非退化情况下，即 P 中不存在四点共圆时，Delaunay 图成为一个三角剖分，称为 Delaunay 三角剖分，用 DT(P)表示。退化情况下的 Delaunay 图可以比较容易地转化为 Delaunay 三角剖分。我们将

Delaunay 三角剖分中的每个三角形称之为 Delaunay 三角形。每个 Delaunay 三角形中的每条边称之为 Delaunay 边，每个顶点是其相对边的 Delaunay 顶点[WangJY2011]。

平面离散点集的 Voronoi 图与其 Delaunay 三角剖分存在如下关系。

（1）每个 Delaunay 三角形对应一个 Voronoi 顶点，其外接圆的圆心是一个 Voronoi 顶点；

（2）Delaunay 三角剖分中的一个顶点（即站点）对应一个 Voronoi 区域；

（3）每条 Delaunay 边对应一条 Voronoi 边。

因此，对任一平面点集 $P=\{p_1, p_2, \cdots, p_n\}$，将其 Delaunay 三角剖分后，可通过如下操作得到 VD (P)。

（1）如果两个三角形有公共边，连接它们外接圆的圆心可得到一条 Voronoi 边。

（2）如果一个三角形的一条边没有相邻的三角形，由其外接圆的圆心开始做一射线，方向与三角形的这一条边垂直，可得到一条半开放的 Voronoi 边。

3. 半平面法

对于平面上的两个点 p_i 和 p_j，用 $h(p_i, p_j)$表示 p_i 所在的半平面。平面上一点 $q \in h(p_i, p_j)$当且仅当$|qp_i| \leqslant |qp_j|$，$j \neq i$，p_i，$p_j \in P$。

一个 Voronoi 区域 VR(p_i)也可以看作是 p_i 与其他所有站点所产生的半平面的交集。因此，可以用半平面交的方法描述每个站点 p_i 的 Voronoi 区域：

$$\mathrm{VD}(P_i) = \bigcap_{\substack{1 \leqslant j \leqslant n \\ j \neq i}} h(p_i, p_j)$$

基于这种定义，对于每一个站点 p_i，我们计算出 p_i 与其他站点的半平面，然后求交集即可得到 VR(p_i)，从而得到整个点集的 Voronoi 图[WangJY2011]。

4. 波传播法

平面离散点集的 Voronoi 图也可以用波传播现象进行描述：在所给的站点 $p_1, p_2, \cdots, p_n$ 处以同样的速度同时发射电波，电波相遇的地方就是 Voronoi 边。这种描述方法也称为森林着火法，即将平面看作一片森林，假设在所给的站点 p_1，p_2，…，p_n 处同时着火，并假设火以同样的速度蔓延，火相遇的地方就是 Voronoi 边。

上述描述也是图像的骨架的定义。一般地，图像的骨架具有连通性、居中性和像素宽度为 1 等重要特性[YangCL2000]。

5. 圆锥投影法

设 $p(x_1, y_1)$ 是三维坐标系的 xy 平面上一点。我们构造一个以 $p(x_1, y_1)$ 为顶点的圆锥 $Co(p)$，它的轴垂直于 xy 平面且侧面以 45° 倾斜。该锥面的方程为：

$$z = \sqrt{(x - x_1)^2 + (y - y_1)^2}$$

对于 xy 平面上的两点 p_1 和 p_2，对应的 $Co(p_1)$ 和 $Co(p_2)$ 相交于一条双曲线，易知该双曲线位于一个垂直于 xy 平面的平面上，且其在 xy 平面上的投影正好是线段 $p_1 p_2$ 的垂直平分线（如图 2.2 所示）。

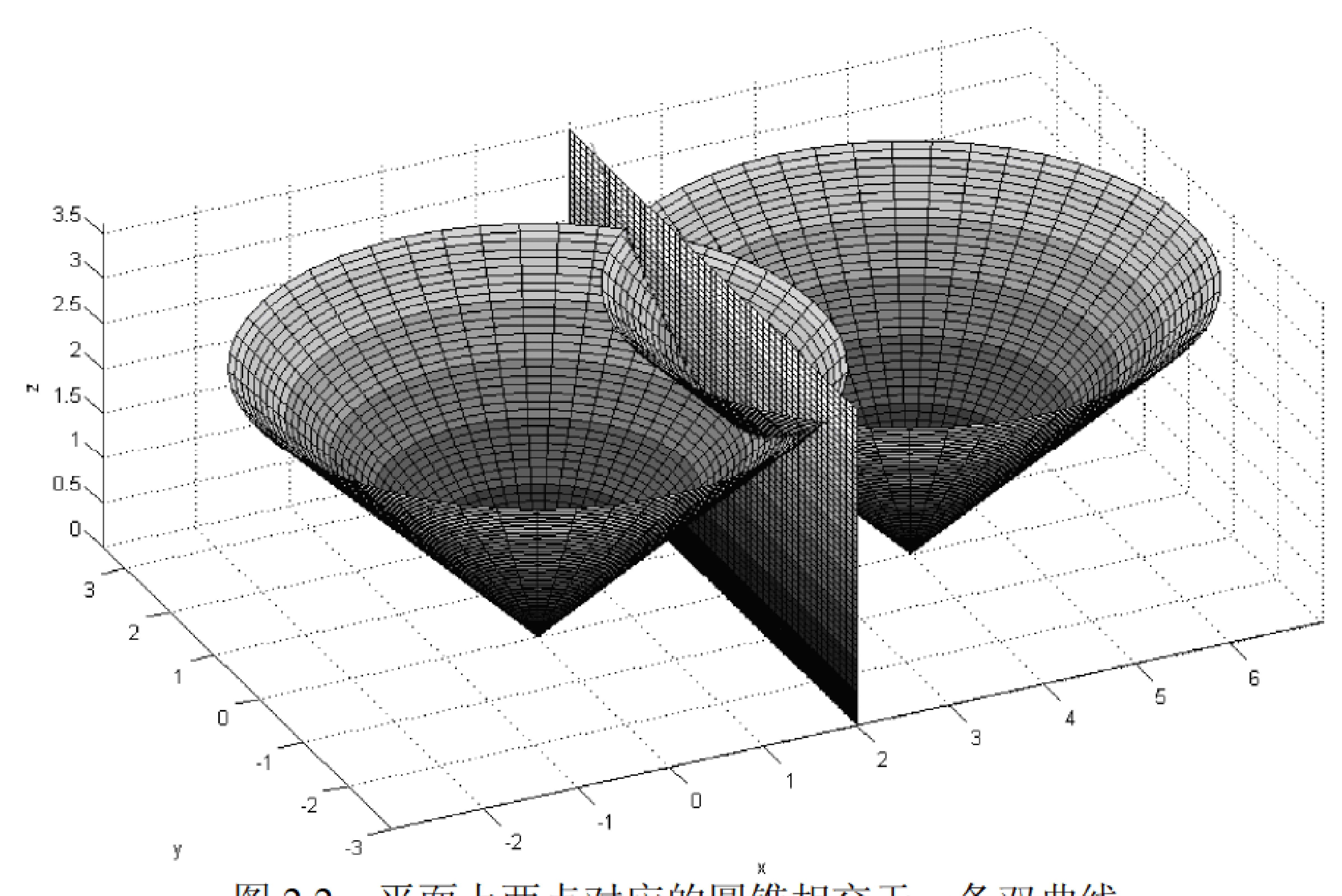


图 2.2　平面上两点对应的圆锥相交于一条双曲线

因此，对于平面离散点集 $P=\{p_1, p_2, \cdots, p_n\}$，可以构造 n 个这样的圆锥 $Co(p_1)$，…，$Co(p_n)$。假设这些圆锥不透明，若从 $z=-\infty$ 方向看去（即沿 $z=-\infty$ 方向平行投影），所看到的圆锥之间的交线（的投影）恰好是 P 的 Voronoi 图（如图 2.3 所示）。图 2.3 也可以看作是波传播法的一个几何视图。

基于这种定义，可以设计扫描线算法构造平面离散点集的 Voronoi 图，具体内容见 2.2.2 节。

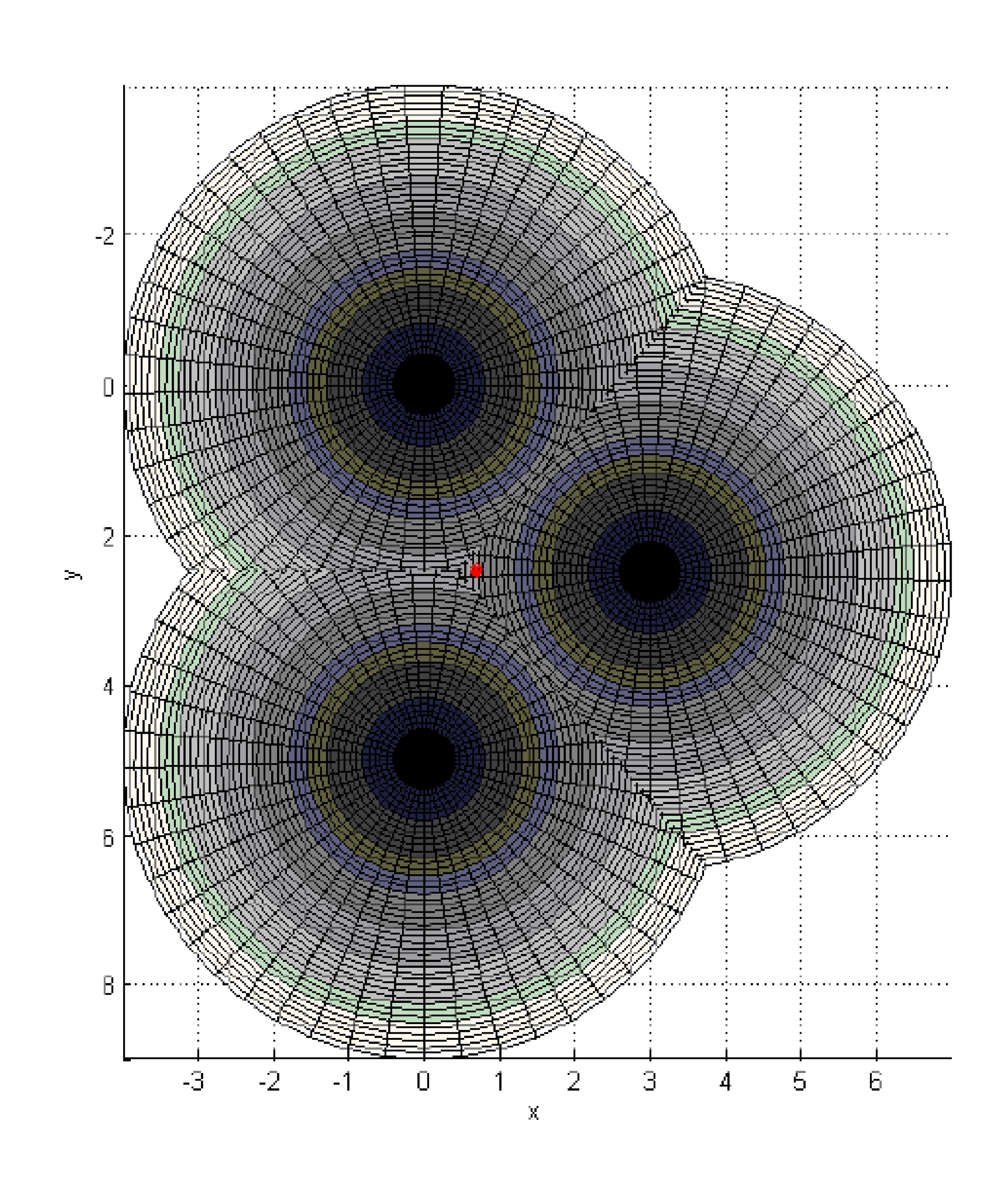


图 2.3　从 $z=-\infty$ 看 $n(n=3)$ 个不透明的圆锥，
看到的圆锥之间的交线的投影恰好是 n 个点的 Voronoi 图

6. 凸包投影法

对于平面离散点集 $P=\{p_1, p_2, \cdots, p_n\}$，可以从另一个角度来观察理解它的 Delaunay 三角剖分和 Voronoi 图，也就是观察它们和三维凸包的关系。

我们首先计算 P 的每个站点 $p_i(x_i, y_i)$ 在抛物面 Π_1：$p_{i3}=x_i^2+y_i^2$ 上的对应点 $p_{i3}(x_i, y_i, x_i^2+y_i^2)$，它是通过 $p_i(x_i, y_i)$ 作垂直线，该垂直线和抛物面 Π_1 的交点。对所有对应点作凸包。在非退化情况下，该凸包的每个面都是三角

形。过其中任意一个三角形△$p_{i3}p_{j3}p_{k3}$（p_{i3},p_{j3},p_{k3} 在 xy 平面的垂直投影分别是站点 $p_i(x_i,y_i)$，$p_j(x_j,y_j)$ 和 $p_k(x_k,y_k)$）的平面 Π_2 有如下特点。

（1）Π_2 与抛物面 Π_1 交于过点 p_{i3},p_{j3},p_{k3} 的椭圆，该椭圆在 xy 平面的垂直投影是一个过 $p_i(x_i,y_i)$、$p_j(x_j,y_j)$ 和 $p_k(x_k,y_k)$ 的圆。

（2）任何其他站点在抛物面 Π_1 上的对应点都在 Π_2 的上方，且任何其他站点都在过 $p_i(x_i,y_i)$、$p_j(x_j,y_j)$ 和 $p_k(x_k,y_k)$ 的外接圆外，即过 $p_i(x_i,y_i)$、$p_j(x_j,y_j)$ 和 $p_k(x_k,y_k)$ 的外接圆是空圆，△$p_ip_jp_k$ 是 Delaunay 三角形。

因此，可以将平面离散点集 P 的 Delaunay 三角剖分看作是所有站点在抛物面 Π_1：$p_{i3}=x_i^2+y_i^2$ 上的对应点的凸包去掉外法向朝上的面后在 xy 平面的垂直投影。

基于上述定义，我们得到计算平面离散点集 P 的 Delaunay 三角剖分和 Voronoi 图的一种方法：首先计算所有站点 $p_i(x_i,y_i)$ 在三维空间上的对应点 $p_{i3}(x_i,y_i,x_i^2+y_i^2)$ 的凸包；然后将该凸包去掉外法向朝上的面后投影到 xy 平面，即可得到 P 的 Delaunay 图。由此容易构造 P 的 Delaunay 三角剖分，由 Delaunay 三角剖分可在 $O(n)$ 时间内构造 P 的 Voronoi 图。由于可在 $O(n\log n)$ 时间内计算三维空间点集的凸包，所以整个算法的时间复杂度为 $O(n\log n)$。

2.1.2 性质

对于点集 $P=\{p_1, p_2, \cdots, p_n\}$，其 Voronoi 图 VD($P$)和 Delaunay 三角剖分 DT($P$)有最邻近性、局部性等性质。下面介绍部分主要性质（此处对各性质的证明从略，请参考文献[WangJY2011]）。

性质 2.1（大小性质）

（1）VD(P)和 DT(P)的边数均小于等于 $3n-6$。

（2）VD(P)的顶点数和 DT(P)的三角形数均小于等于 $2n-5$。

（3）任何一个 Voronoi 区域所包含的 Voronoi 边数的平均数小于 6。

性质 2.1 可用于与 Voronoi 图 VD (P)和 Delaunay 三角剖分相关的算法的复杂度分析。根据性质 2.1，可用 $O(n)$ 空间存储 Voronoi 图和 Delaunay 三角剖分的结果。

性质 2.2（空圆性质）

（1）设 p_1、p_2 和 p_3 是 VD (P)中一个 Voronoi 顶点 v 所关联的三个站点，则以 v 为圆心经过 p_1，p_2 和 p_3 三个站点的圆是一个空圆，即该圆中不存在 P 的其他站点。

（2）对于站点 p_i 和 p_j ($i \neq j$)，$p_i p_j$ 是 DT(P)的一条边当且仅当存在一个仅过 p_i 和 p_j 的闭圆盘，该圆盘为空圆。因此亦可得：对于 P 的一个站点 p_i，p_i 和其最近的站点 p_j 之间必有一条 DT(P)的边；站点 p_i 和 p_j 是 DT(P)中距离最短的边，当且仅当 p_i 和 p_j 是 P 中所有站点中距离最近的一对点。

（3）站点 p_i、p_j、p_k 为 DT(P)中的一个 Delaunay 三角形的顶点，当且仅当存在一个经过三点 p_i、p_j、p_k 的圆，该圆内部不含任何站点。其等价于：对任何给定四点，用 Delaunay 三角剖分得到的两个三角形一定是最小角最大。

对于任意一点 q，性质 2.2 可用于设计在 P 中查找其最近点的算法：在 VD (P)中，如果 q 落在站点 p_i 的 Voronoi 区域内，则 p_i 是 P 中离 q 最近的站点；在 DT(P)中，如果 q 落在站点 p_i、p_j、p_k 组成的 Delaunay 三角形内，则 p_i、p_j、p_k 中离 q 最近的即为 P 中离 q 最近的站点。

性质 2.2 也可用于设计查找 P 中最近点对的算法：对于任意一个点集 P，首先计算 DT(P)，然后查找 DT(P)最短的边，即可得到 P 中距离最近的点对。

性质 2.2 表明 VD (P)和 DT(P)具有局部性质：插入或删除一个站点，只会影响 VD (P)和 DT(P)的一部分，这是逐点插入法等构造 VD (P)和 DT(P)的重要基础（见 2.2 节中的构造方法）。

性质 2.3（凸包性质）

（1）将 DT(P)中所有只有一个相邻三角形的边依次连接得到的是点集 P 的凸包。

（2）一个站点 p_i 的 Voronoi 区域 VR(p_i)是开放的，当且仅当 p_i 位于点集 P 的凸包上。

2.2 构 造 方 法

目前已有很多离散点集的 Voronoi 图的构造算法：半平面法、分而治之法、逐点插入法以及扫描线法等。另外，如 2.1 节所述，也可先计算所给站点的 Delaunay 三角剖分，然后再计算 Voronoi 图[WangJY2011]。根据实现过程，Delaunay 三角剖分的算法则有分而治之算法、逐点插入法和三角网生长法等。下面主要介绍几种实用的生成 Voronoi 图和 Delaunay 三角剖分的方法。

2.2.1 逐点插入法生成 Voronoi 图

逐点插入法是最常用的构造 Voronoi 图的算法之一[Green1978]。其基本思想是：逐步加入站点，在已构造的 Voronoi 图的基础上利用局部特性，通过局部修改，快速生成新的 Voronoi 图。即对于平面离散点集 $P=\{p_1, p_2, \cdots, p_n\}$，在生成点集 $P_k=\{p_1, p_2, \cdots, p_k\}(k<n)$ 的 Voronoi 图的基础上，加入 p_{k+1}，并查询新加入站点 p_{k+1} 在 VD (P_k)中的位置，然后通过局部修改来构造 $P_{k+1}=\{p_1, p_2, \cdots, p_{k+1}\}$的 Voronoi 图。如此不断加入新的站点，最后可得到 VD (P)。p_{k+1} 加入后，在其周围形成一个离 p_{k+1} 最近的点组成的 Voronoi 区域 VR(p_{k+1})。它是从附近的 Voronoi 区域中分割出来的（如图 2.4（a）所示）。因 p_{k+1} 落在 VR(p_i)中，分割就从 VR (p_i)开始。用 p_{k+1} 和 p_i 的垂直平分线把 VR(p_i)分成两部分，每部分的点分别离 p_{k+1} 和 p_i 最近。它和 VR(p_i)的边界交于 t 和 q 两点。在 q 点处开始进入 VR(p_j)，这时 VR(p_j)中也有一部分点和 p_{k+1} 最近。因此用 p_{k+1} 和 p_j 的垂直平分线把 VR(p_j)分成两部分。该线交 VR(p_j)的边界交于 r 和 q 两点。用 r 代替 q 重复上述工作，直到有一条垂直平分线通 t 点，形成一个多边形为止（如图 2.4（a）中的虚线多边形所示）。最后把该多边形中间的边和顶点删除，便形成 p_{k+1} 的 Voronoi 多边形 VR(p_{k+1})（如

图 2.4（b）所示）。

算法 2.1　*CreateVD-increment* (*P*) —逐点插入法生成 Voronoi 图

输入：平面离散点集 $P=\{p_1, p_2, \cdots, p_n\}$。

输出：VD (*P*)。

（1）如果 $n \leqslant 1$，返回。

（2）计算 VR(p_1)和 VR(p_2)。//通过计算 p_1、p_2的中垂线得到

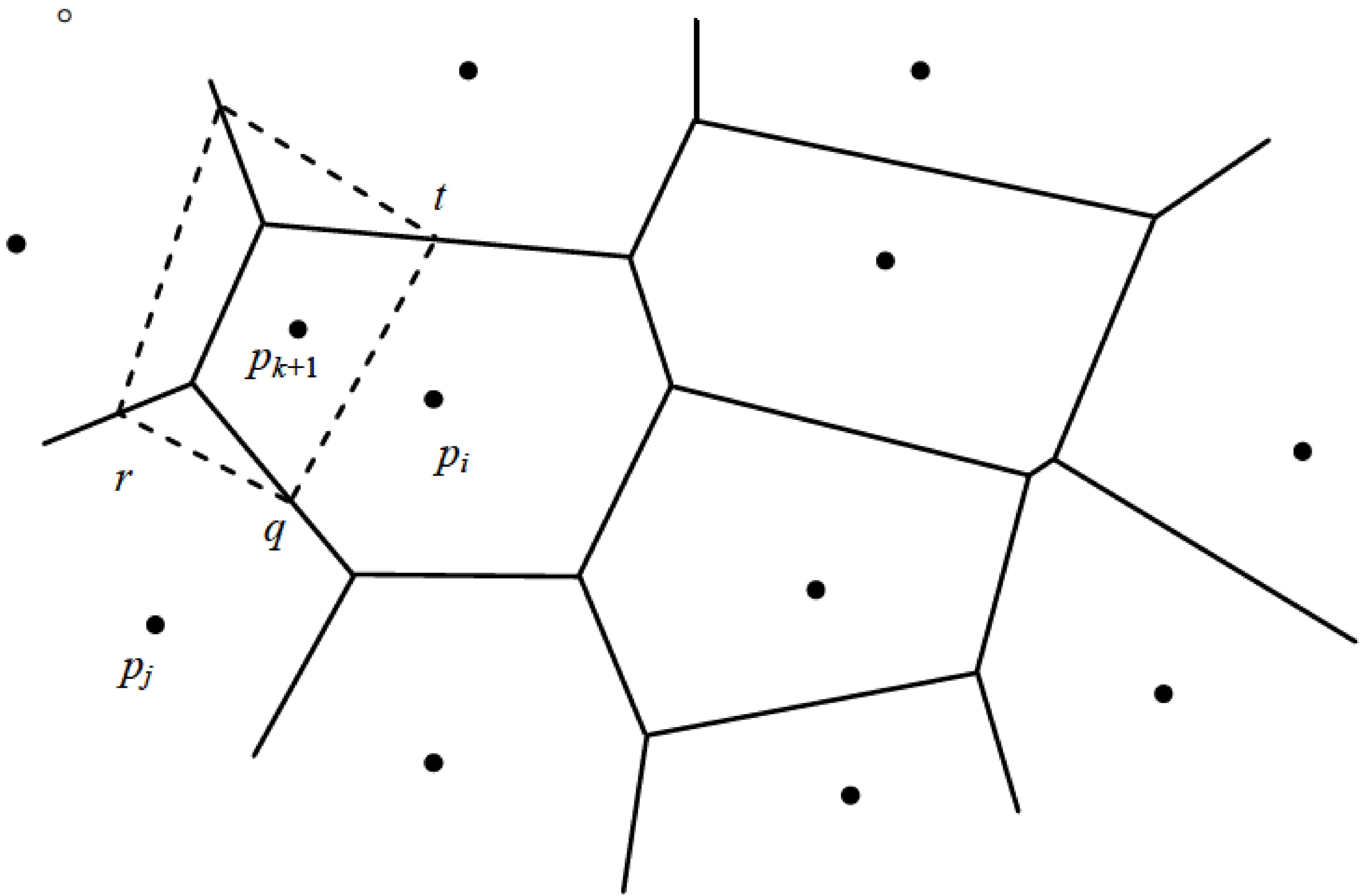
（3）**for**($k=3$；$k \leqslant n$；$k=k+1$){

（4）计算 p_k所属的 Voronoi 区域，设为 VR(p_i)。

（5）计算 p_k的 Voronoi 区域 VR(p_k)。

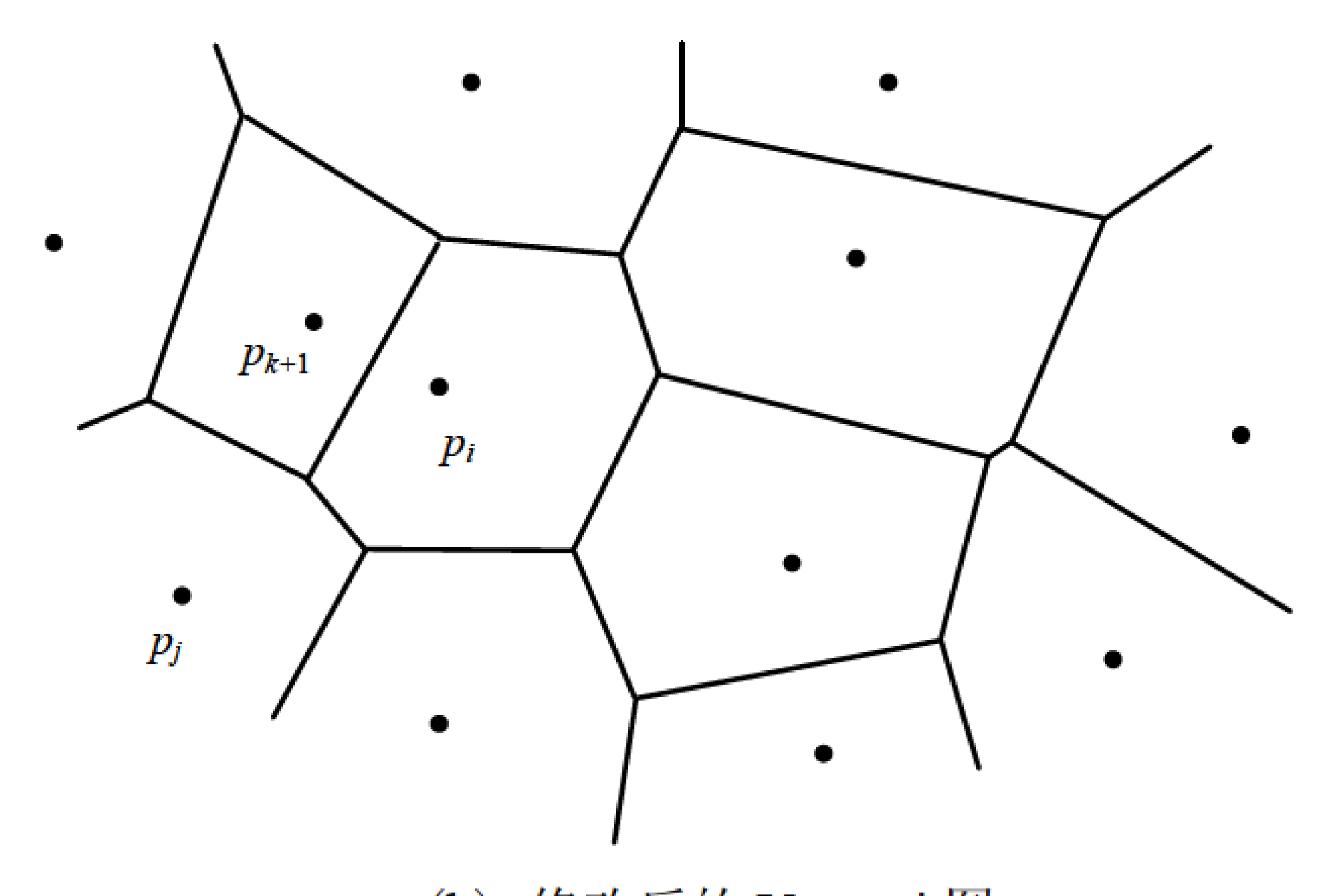
（6）}

实际上，该算法中新加入一个点的定位，最坏情况下估计花费时间为 $O(n)$。一般情况下计算 p_k 的 Voronoi 多边形和修改相应站点的 Voronoi 多边形可在常数时间内完成，最坏情况下最多需要 $O(n)$时间，所以插入一个站点、计算新的 Voronoi 图需要 $O(n)$时间。算法总的时间复杂度为 $O(n^2)$。虽然算法的时间复杂度较高，但因其简单且易于理解与编程，所以是最常用的算法之一。



(a) 新加入站点的定位与其 Voronoi 区域计算

图 2.4　加入一个新点 p_{k+1} 后更新 Voronoi 图的过程



(b) 修改后的 Voronoi 图

图 2.4　（续）

2.2.2　扫描线法生成 Voronoi 图

根据 2.1.1 节所介绍的圆锥投影法的原理，可以设计基于扫描线的算法来构造离散点集的 Voronoi 图[de Berg 2000，Fortune1987，Guibas1988]。

由 2.1.1 节知，对于平面离散点集 $P=\{p_1, p_2, \cdots, p_n\}$，可以构造 n 个圆锥 $Co(p_1)$，…，$Co(p_n)$（$Co(p_i)$以 p_i 为顶点，它的轴垂直于 xy 平面且侧面以 45° 倾斜）。假设这些圆锥不透明，若从 $z=-\infty$ 方向看去（即沿 $z=-\infty$ 方向平行投影），所看到的圆锥之间的交线（的投影）恰好是 P 的 Voronoi 图。

这里，我们设置一个与 xy 平面倾角为135° 且平行于 y 轴的平面 π（如图 2.5 所示），其与 xy 平面的交线为 L。从左往右沿 x 轴逐步移动 π（即 L）。在 L 到达 p 点前，π 与 $Co(p)$ 不相交；当 L 到达 p 点时，π 相切于 $Co(p)$ 的左侧；当 L 到达 p 的右侧，π 与 $Co(p)$ 总相交于一条抛物线，其在 xy 平面上

的投影恰好是点 p 和直线 L 的中分线（是一条抛物线）$B(p,L)$，即 $B(p,L)$ 上的任一点到 p 和 L 的距离都相等。

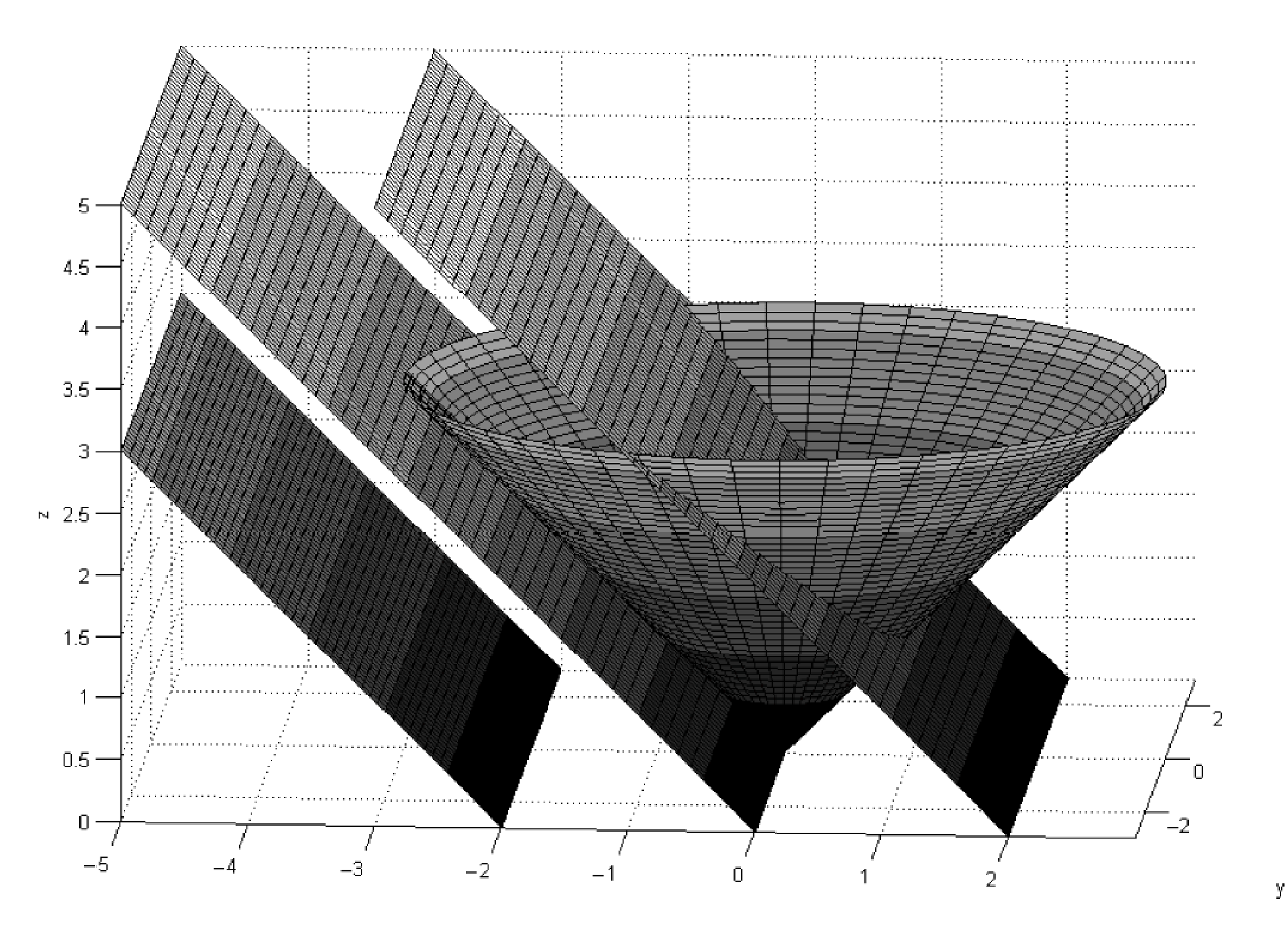


图 2.5　倾角为 135°且平行于 y 轴的平面与站点及其圆锥的位置关系

对于三维坐标系的 xy 平面上两点 p_1 和 p_2，当 p_1 和 p_2 都在 L 的左侧时，π 分别与 $Co(p_1)$，$Co(p_2)$ 相交于两条抛物线；如果这两条抛物线有交点 q，q 必在 $Co(p_1)$ 和 $Co(p_2)$ 交线上，即 q 的投影必在线段 p_1p_2 的垂直平分线上（如图 2.6 所示）。因此，对于平面离散点集，当 π（即 L）从左往右沿 x 轴逐步移动时，在每个固定时刻，位于 L 左侧的所有站点和直线 L 的 Voronoi 图（站点和直线间的 Voronoi 边就是到站点和直线的垂直距离相等的点的轨迹，该轨迹是一条抛物线）就是我们从 $z=-\infty$ 方向看到的结果（假设所有圆锥和 π 不透明）。L 右侧的站点则对其不产生影响（被 π 完全遮挡了）。

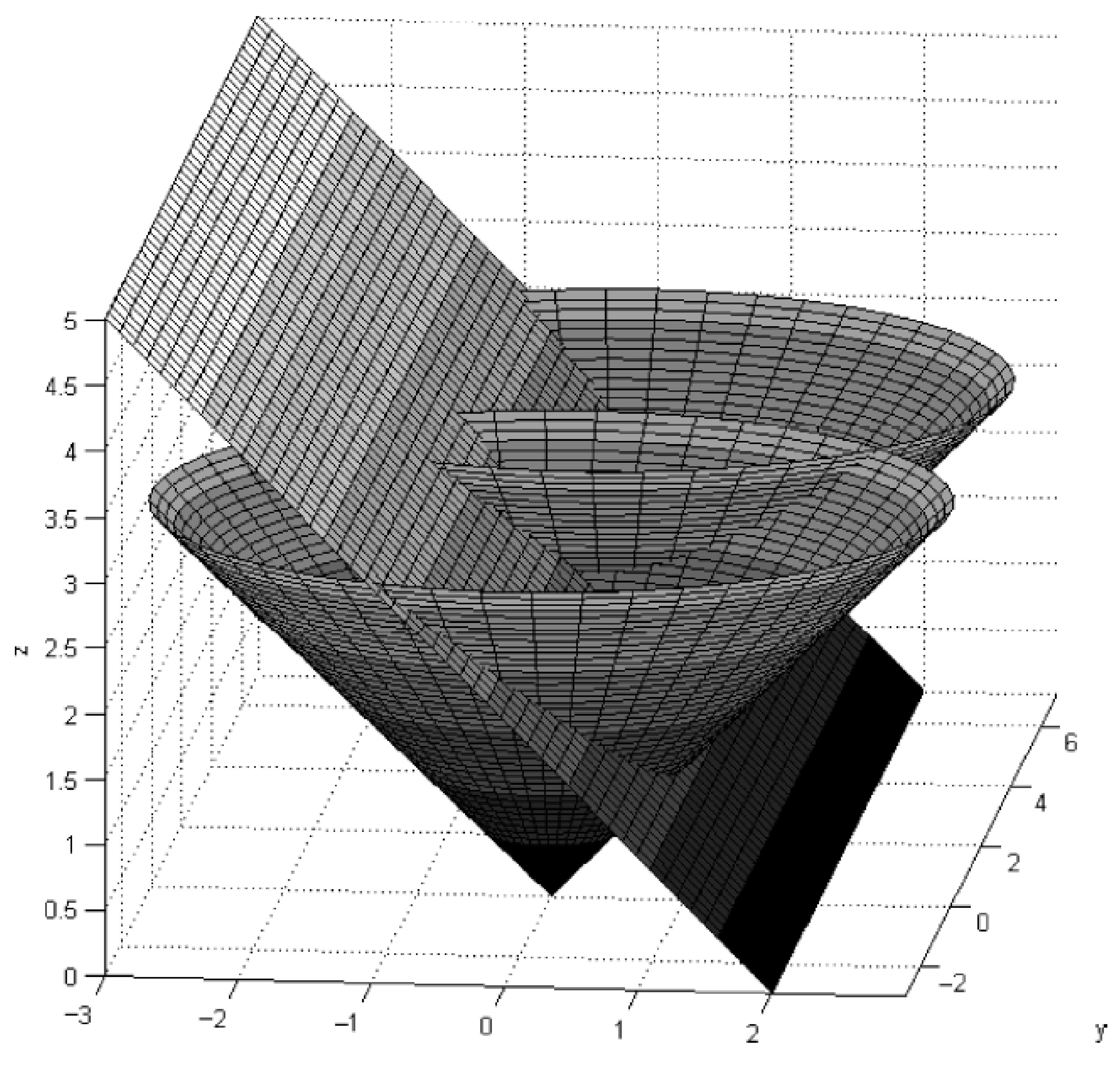


图 2.6　平面π与两个圆锥相交的情况图

其中 L 的 Voronoi 区域的边界是由一组抛物线弧依次首尾连接成的一条曲线（两端的两条抛物线弧各有一端是无限的），位于 L 和其左侧的站点 p_1，p_2，…，p_i 之间，是关于 L 严格单调的，且随着 L 的向右移动而逐渐移动和变动，因此称为海岸线（如图 2.7 所示）。组成海岸线的那些抛物线弧首尾依次相连，其接合点称为断点。如前所述，每个断点都落在 Voronoi 图的某条边上。

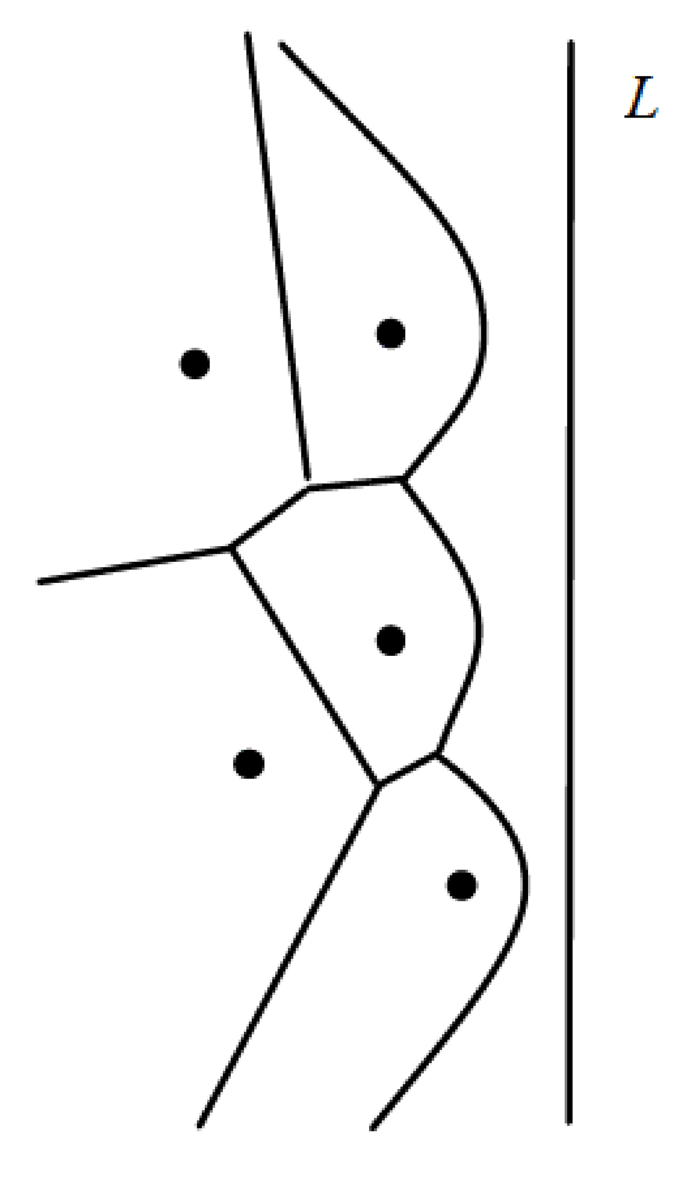


图 2.7　直线 L 与其左侧站点的 Voronoi 图

由上面的观察可知，对于平面离散点集 $P=\{p_1, p_2, \cdots, p_n\}$，我们用平行于 y 轴的扫描线 L 从左到右依次扫描，扫描过程中海岸线随着 L 的向右移动而逐渐向右移动并不断变化，新断点也随之不断生成。由于海岸线的每段弧是 L 与其左侧某个站点 p 的中分线，所以对于海岸线左侧的任意一点 v，必存在至少一个站点 p，使得 v 到 p 的距离小于 v 到 L 的距离，当然也小于 v 到 L 右侧所有站点的距离，即 v 必属于 L 左侧某个站点的 Voronoi 区域。所以对于当前扫描线，已得到的海岸线左侧的 Voronoi 图就是整个点集的 Voronoi 图（在海岸线左侧）的一部分，它们是已得到的所有断点的轨迹（该部分加上海岸线就是直线 L 和其左侧的所有站点的 Voronoi 图），不受 L 右侧站点的影响。随着 L 向右移动，新的站点不断加入，海岸线左侧的 Voronoi 图不断生长，最后即可得到整个点集的 Voronoi 图。这就是平面扫描法构造 Voronoi 图的基本思想，算法设计的具体细节请参阅文献[de Berg 2000，Fortune1987，Guibas1988，WangJY2011]。

扫描线算法可以视为一种逐点插入法，可在 $O(n\log n)$ 时间和 $O(n)$ 空间内构造出整个 Voronoi 图。该算法的缺点是所需的数据结构比较复杂。

2.2.3 基于扫描线的逐点插入法生成 Voronoi 图

本节介绍一种将扫描线法和逐点插入法相结合生成 Voronoi 图的算法[Meng2010]。该算法以扫描线的方式从左向右逐个扫描点集中的站点，并在扫描过程中以增量的方式生成所有已扫描过的站点的 Voronoi 图。对于当前扫描到的站点的定位，算法引入了右凸链概念，支持简单、快速的定位。

1. 右凸链

设 $Q=\{q_1, q_2, \cdots, q_n\}$ 为一个凸多边形，l 是位于 Q 右侧的一条垂直扫描线（如图 2.8 所示）。Q 的外部 Voronoi 图（参见第 3 章）如图 2.8 中虚线部分所示，是一个特殊的 Voronoi 图：其 Voronoi 边位于多边形的外部，且垂直于和其相关联的 Q 的边；Q 的一条边或一个顶点的 Voronoi 区域只有两条边，该区域是开放的。

需要注意的是，这里提到两类 Voronoi 图，一类是我们要构造的点集的 Voronoi 图，在构造过程中会不断生成新的 Voronoi 图；另一类是凸多边形的

Voronoi 图，用于点的定位，起辅助作用。这里，凸多边形是当前已扫描站点集合的凸包，其上面的点 p_i 的外部 Voronoi 区域用 $VR_o(p_i)$表示。p_i 在 VD (P) 中的 Voronoi 区域则用 $VR_p(p_i)$表示。

在凸多边形 Q 中，我们将那些外部 Voronoi 区域和扫描线 l 相交的顶点和边构成的多边形链称为关于 l 的右凸链（Right Convex Hull Chain，RCHC）。如图 2.8 所示，多边形链 $q_3q_4q_5$ 为关于扫描线 l 的一条右凸链。显然，右凸链关于 l 是严格单调的，即垂直于 l 的任一直线与右凸链只有一个交点。

2. 点定位

前面定义的 RCHC 主要用于快速定位新扫描到的站点 q 的位置，即在已扫描站点集合的 Voronoi 图中，q 属于哪一个 Voronoi 区域。由于 RCHC 的顶点或边的外部 Voronoi 区域的边是依次和 l 相交的（如图 2.8 所示），所以可以用二分法找到一个顶点 p_i（或一条边 p_ip_{i+1}），q 包含在 $VR_o(p_i)$（或 $VR_o(p_i p_{i+1})$）中。这个过程的用时为 $O(\log n)$。如图 2.9 中的多边形链 $p_1p_2\ldots p_6$ 是关于扫描线 l 的右凸链，q 是 l 上的一点，由于 q 在 $p_1p_2\ldots p_6$ 的中点 p_3 的外部 Voronoi 区域 $VR_o(p_3)$的边的下方，因此接下来只须继续用二分法搜索多边形链 $p_1p_2p_3$，最后即可发现 q 位于 $VR_o(p_2)$中。

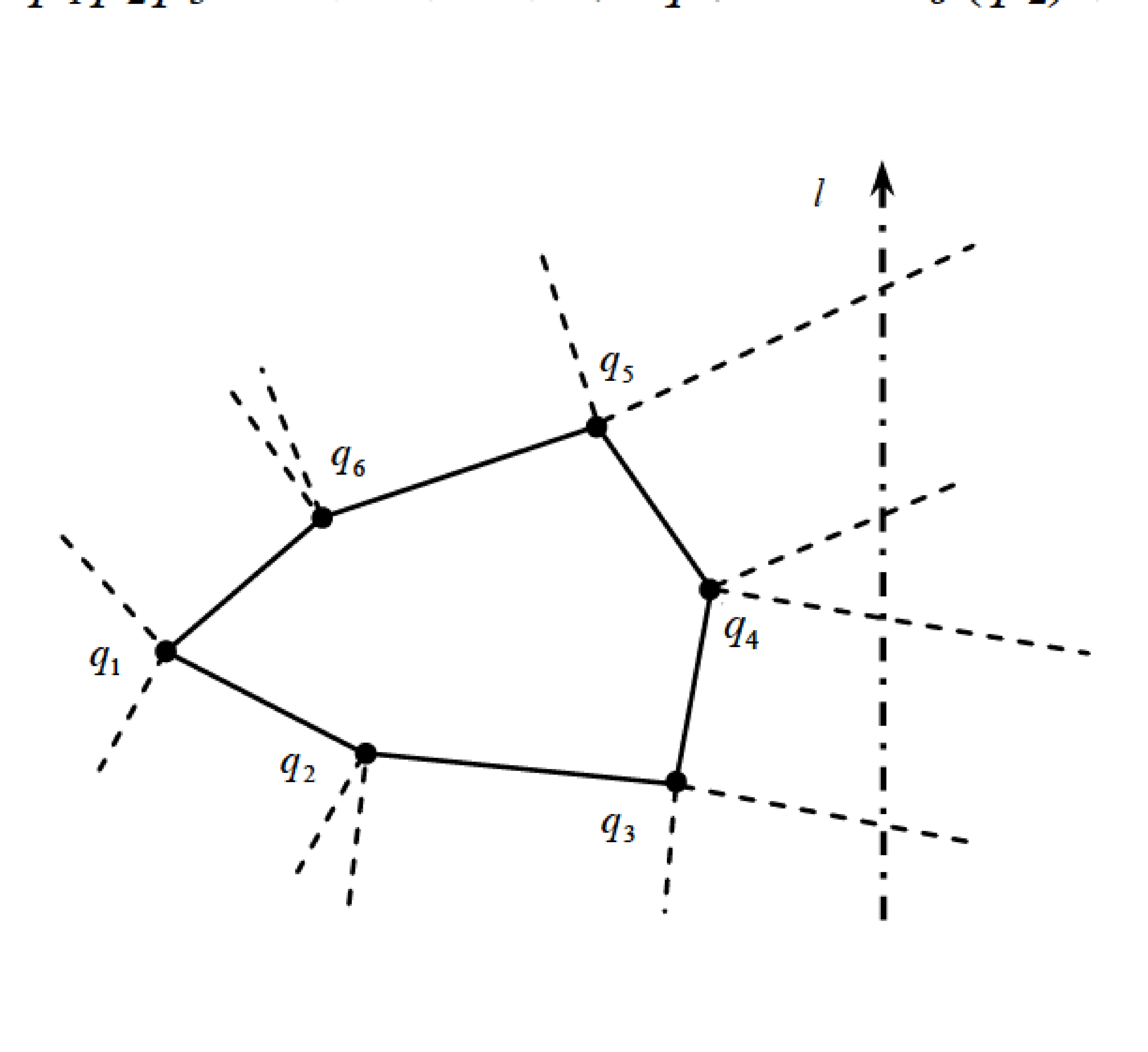


图 2.8　$Q=\{q_1, q_2, \ldots, q_6\}$的关于 l 的右凸链

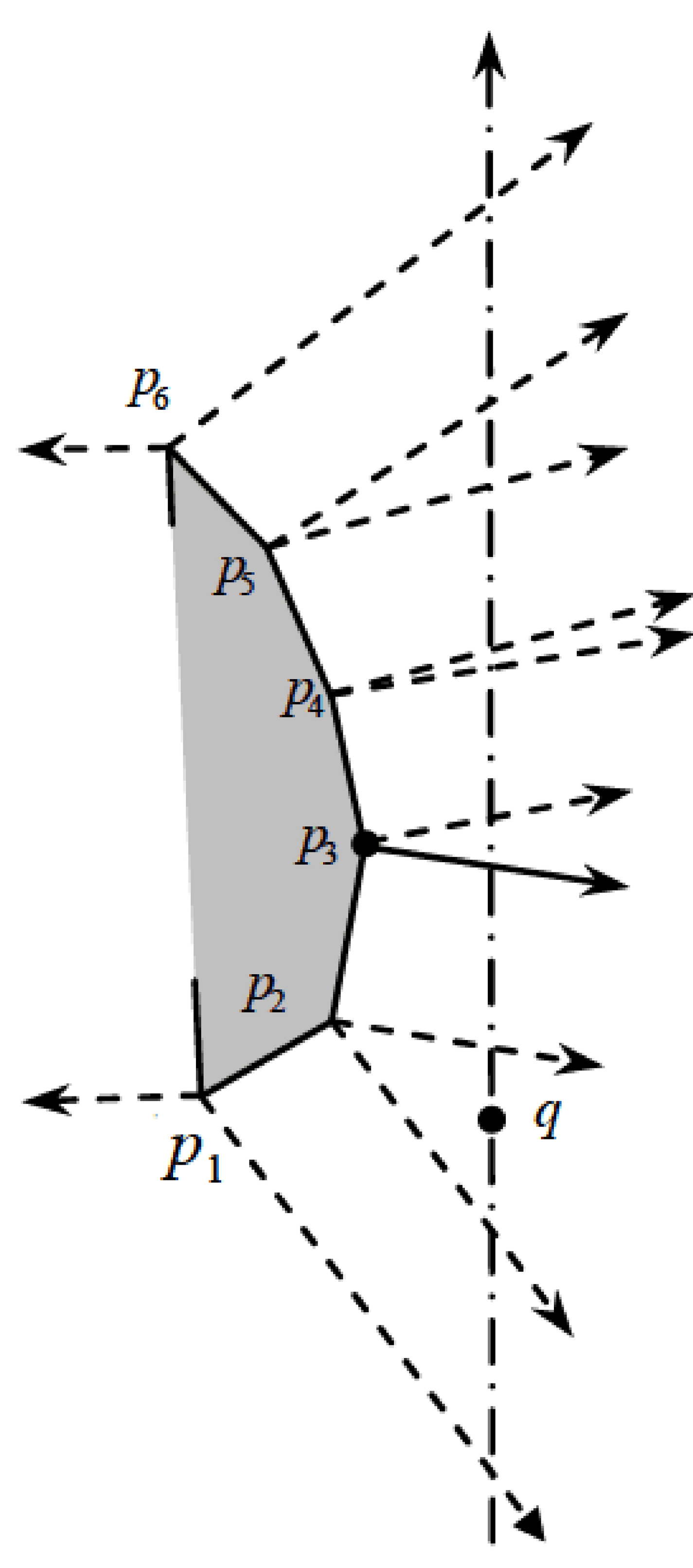


图 2.9　基于右凸链查找包含 q 的 Voronoi 区域

下面需要证明的是，如果 q 位于 $VR_o(p_i)$中，则 q 必位于 $VR_p(p_i)$中。

假设点集 $P=\{p_1, p_2, \cdots, p_n\}$位于垂直线 l 的左侧，其 Voronoi 图 VD (P)已构造出来。另设点 v_1，v_2，…，v_m ($m<n$)是 P 的凸包上的顶点，且多边形链 $v_1v_2\ldots v_m$ 是关于垂直扫描线 l 的右凸链；q 是 l 上一点。

定理 2.1 如果点 q 位于 $VR_o(v_j)(1\leqslant j\leqslant m)$中，则 q 必位于 $VR_p(v_j)$中。

证明：在 RCHC $v_1v_2\ldots v_m$ 的外部 Voronoi 图中，由于 q 位于 $VR_o(v_j)$中，所以 v_j 是 RCHC $v_1v_2\ldots v_m$ 中离 q 最近的点。

由于 v_{j-1}，v_j，v_{j+1} 是 $v_1v_2\ldots v_m$ ($m\leqslant n$)上 3 个连续的顶点，$v_1v_2\ldots v_m$ 是点集 P 的凸包的一部分，所以在整个 VD (P)中，点 v_{j-1} 与 v_j、v_j 与 v_{j+1} 之间都存在 Voronoi 边。其中，v_{j-1} 与 v_j 之间的 Voronoi 边与 $VR_o(v_j)$的一条垂直于 $v_{j-1}v_j$ 的 Voronoi 边平行，v_{j+1} 与 v_j 之间的 Voronoi 边与 $VR_o(v_j)$的一条垂直于 $v_{j+1}v_j$ 的 Voronoi 边平行。由于 v_j 是 P 的凸包的一个顶点，所以 $VR_p(v_j)$是开放的，因此可知 q 必位于 $VR_p(v_j)$中（如图 2.10（b）所示）。证毕。

一旦确定 q 位于 $VR_p(v_j)$中，根据性质 2.2 可得定理 2.2。

定理 2.2 如果点 q 位于 $VR_p(v_j)$ ($1\leqslant j\leqslant m$)中，则在 VD ($P\cup\{q\}$)中，q 和 v_j 之间存在 Voronoi 边。

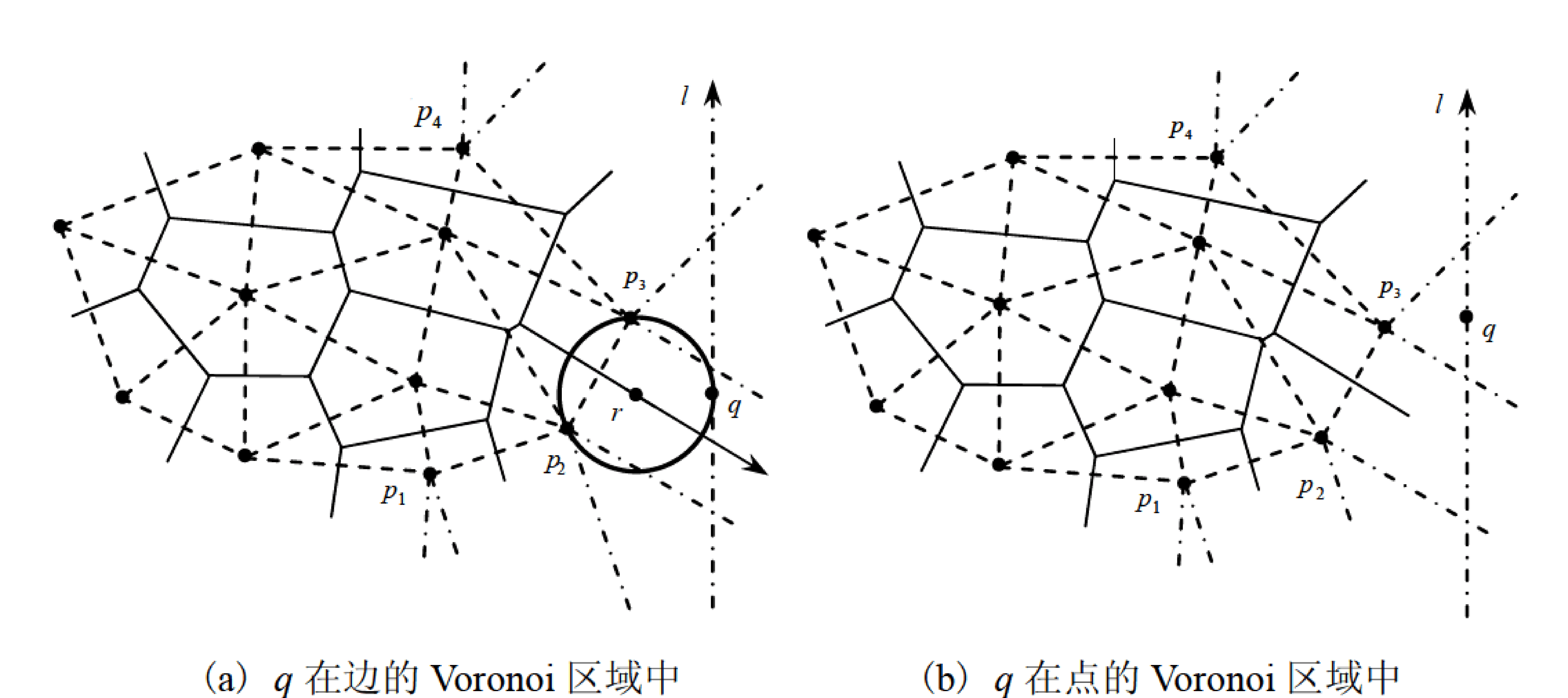
定理 2.3 如果点 q 位于 $VR_o(v_j v_{j+1})$ ($1\leqslant j\leqslant m$)中，则在 VD ($P\cup\{q\}$)中，q 和点 v_j 、q 和点 v_{j+1} 之间分别存在 Voronoi 边。

证明：设$\triangle v_i v_j v_{j+1}$ 是 VD (P)的对偶图 DT(P)中的一个三角形，垂直扫描线 l 上的 q 位于 $VR_o(v_j v_{j+1})$中。如果 q 在$\triangle v_i v_j v_{j+1}$ 的外接圆内，那么为了得到 DT($P\cup\{q\}$)，就需要将 $v_j v_{j+1}$ 用 $v_i q$ 替代，用来构造两个新的三角形，才能满足 Delaunay 的最大空圆原则，即每个三角形的外接圆内不含其他站点。qv_j 与 qv_{j+1} 是 Delaunay 三角剖分 DT($P\cup\{q\}$)中的两条边，因此，它们在 VD ($P\cup\{q\}$)中，q 与 v_j 、q 与 v_{j+1} 之间分别存在 Voronoi 边。否则，如

果 q 在$\triangle v_i v_j v_{j+1}$的外接圆外部，设点 r 是$\triangle v_i v_j v_{j+1}$的外接圆的圆心，则点 r 必位于线段 $v_j v_{j+1}$ 的垂直平分线上。如果沿 $v_j v_{j+1}$ 的垂直平分线拉动点 r 朝凸包外的方向运动，则以点 r 为圆心、过点 v_j 和点 v_{j+1} 两个顶点的圆逐渐变大，且保持空圆特性（即该圆内部没有其他站点）直到遇到 q（如图 2.10（a）所示）。这时，由于 qv_j 和 qv_{j+1} 必是 DT($P\cup\{q\}$)的两条边，因而在 $P\cup\{q\}$ 的 Voronoi 图中，q 和 v_j，q 和 v_{j+1} 之间分别存在 Voronoi 边。证毕。

在图 2.10 中，多边形链 $p_1p_2p_3p_4$ 是关于直线 l 的一条右凸链，q 是 l 上一点。图 2.10（b）的 q 在 $VR_o(p_3)$中，也必在 $VR_p(p_3)$中。p_3q 是 DT($P\cup\{q\}$)中的一条 Delaunay 边，即在 VD ($P\cup\{q\}$)中，p_3、q 之间存在一条 Voronoi 边。图 2.10（a）的 q 在 $VR_o(p_2p_3)$中。这时，p_2q 和 p_3q 都是 DT($P\cup\{q\}$)中的 Delaunay 边，即在 VD ($P\cup\{q\}$)中，点 p_2 和 q、点 p_3 和 q 之间都存在 Voronoi 边。

因此，对于新扫描的一个站点 q，可以先基于右凸链在 $O(\log n)$时间内定位点 q 所在的 Voronoi 区域，并找到一条属于 VD ($P\cup\{q\}$)中的、与 q 相关的 Voronoi 边；然后采用 2.2.1 节中介绍的方法，执行局部修改操作，计算出 q 的 Voronoi 区域，即可得到 VD ($P\cup\{q\}$)。



（a） q 在边的 Voronoi 区域中　　　（b） q 在点的 Voronoi 区域中

图 2.10　q 属于不同类型站点的 Voronoi 区域

3. 更新右凸链

设当前插入点 q 落入 $VR_p(p_i)$中，并已经计算出 VD $(P\cup\{q\})$，则根据前面的分析，qp_i 为 DT$(P\cup\{q\})$中的一条边，即 q、p_i 之间存在 Voronoi 边。计算出的 q 的 Voronoi 区域肯定是开放的，且有两条开放的 Voronoi 边。设这两条开放的 Voronoi 边分别是 q 和 p_j、 q 和 p_k 之间的 Voronoi 边，容易证明，q 与点集 P 的凸包的切线必相切于 p_j 和 p_k （如图 2.11 中的点 p_2 和 p_4）。因此，算法在计算 VD $(P\cup\{q\})$时就可计算出新的右凸链。

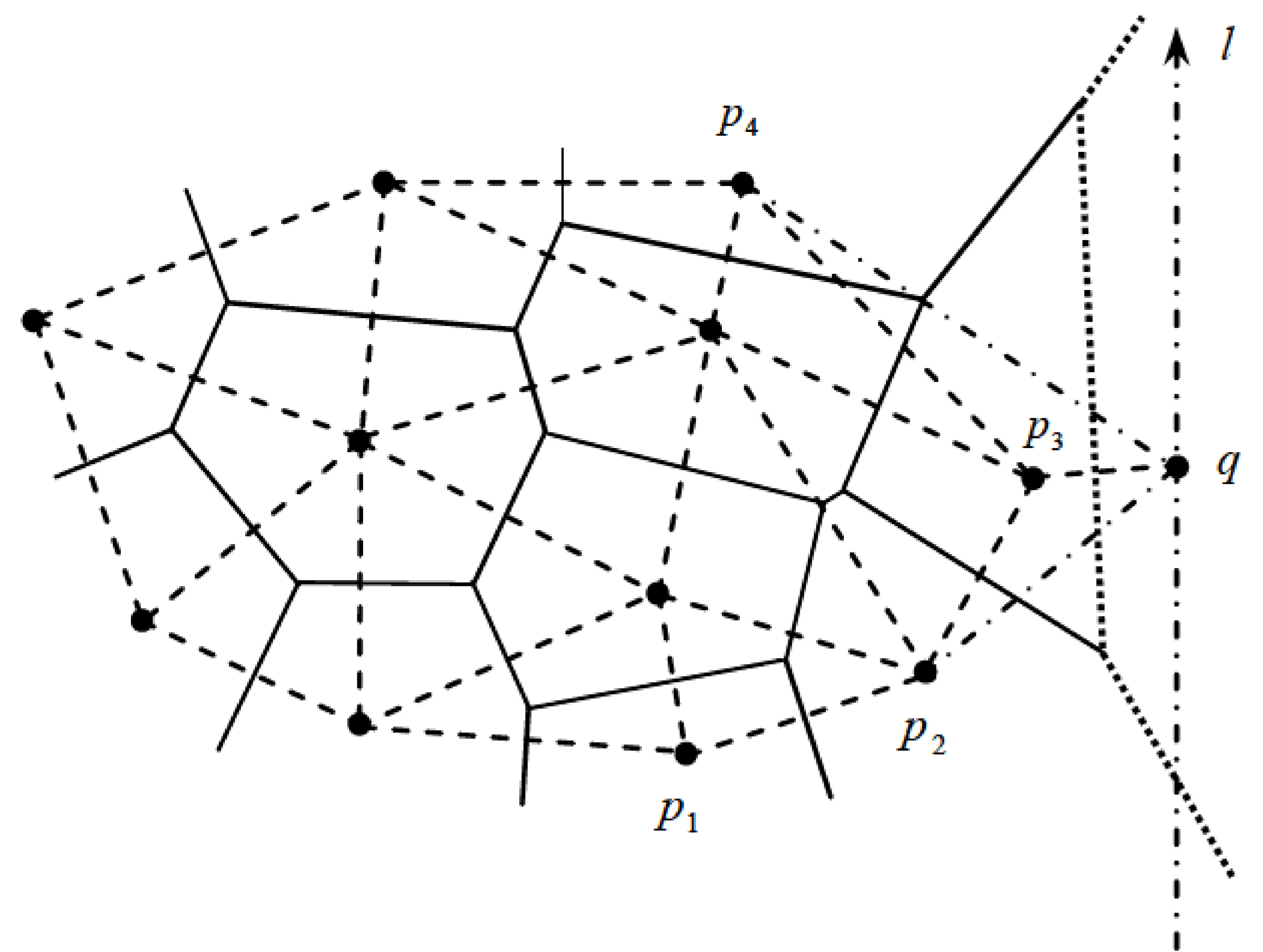


图 2.11　右凸链原为 $p_1p_2p_3p_4$，更新后为 p_2qp_4

4. 算法描述

算法首先对给定的站点按 x 方向由小到大排序，如果方向 x 相同则按 y 方向排序，得到点 p_1，p_2，…，p_n；然后取点 p_1、p_2、p_3，构造它们的 Voronoi 图，并计算初始的右凸链 RCHC；接着依次插入后面的每一个站点 p_k，并基于 RCHC 计算包含点 p_k 的 Voronoi 区域。设 p_k 落在 $VR_p(p_i)$中，则 p_kp_i 为 DT$(P\cup\{p_k\})$的一条边；最后，计算 VD$(P\cup\{p_k\})$，并更新 RCHC。当所有站点都处理完后，即得到整个输入站点集的 Voronoi 图。

算法 2.2　CreateVD (P)——基于扫描线的逐点插入法生成 Voronoi 图

输入：含有 $n\,(n\geqslant 3)$(n)个平面离散点的集合 P。

输出：VD (P)。

（1）如果 $n<3$，则返回。

（2）对给定的站点按 x 方向由小到大排序，如果方向 x 相同则按 y 方向排序，得到点集 p_1，p_2，…，p_n。

（3）构造点 p_1、p_2、p_3 的 Voronoi 图。

（4）计算初始的右凸链 RCHC。// RCHC 有可能是其中的 3 个点或 2 个点。

（5）**for** (k =4；$k\leqslant n$；k= k +1) {

（6）基于 RCHC 用二分法查找包含点 p_k 的 Voronoi 区域。

（7）如果 p_k 在 $\mathrm{VR}_o(v_i)$中，则计算 p_k 与点 v_i 之间的 VD $(P\cup\{p_k\})$的 Voronoi 边。其中，v_i 为 RCHC 的一个顶点。

（8）如果 p_k 在 $\mathrm{VR}_o\,(v_iv_{i+1})$中，计算 p_k 与点 v_i、p_k 与点 v_{i+1} 之间的 VD $(P\cup\{p_k\})$的 Voronoi 边。

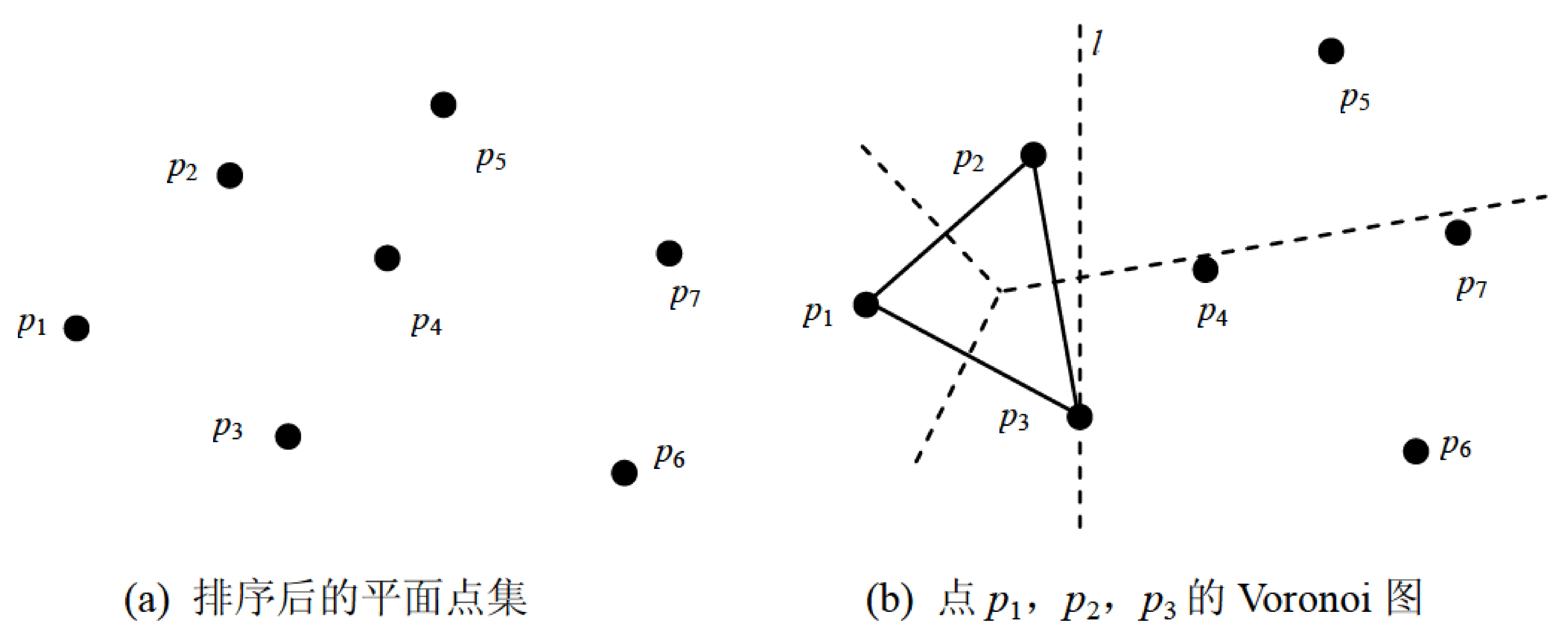
（9）根据上面的 Voronoi 边，采用 2.2.1 节中介绍的方法计算 $\mathrm{VR}_p\,(p_k)$，并计算得到 VD $(P\cup\{p_k\})$。

（10）计算新的 RCHC。

（11） }

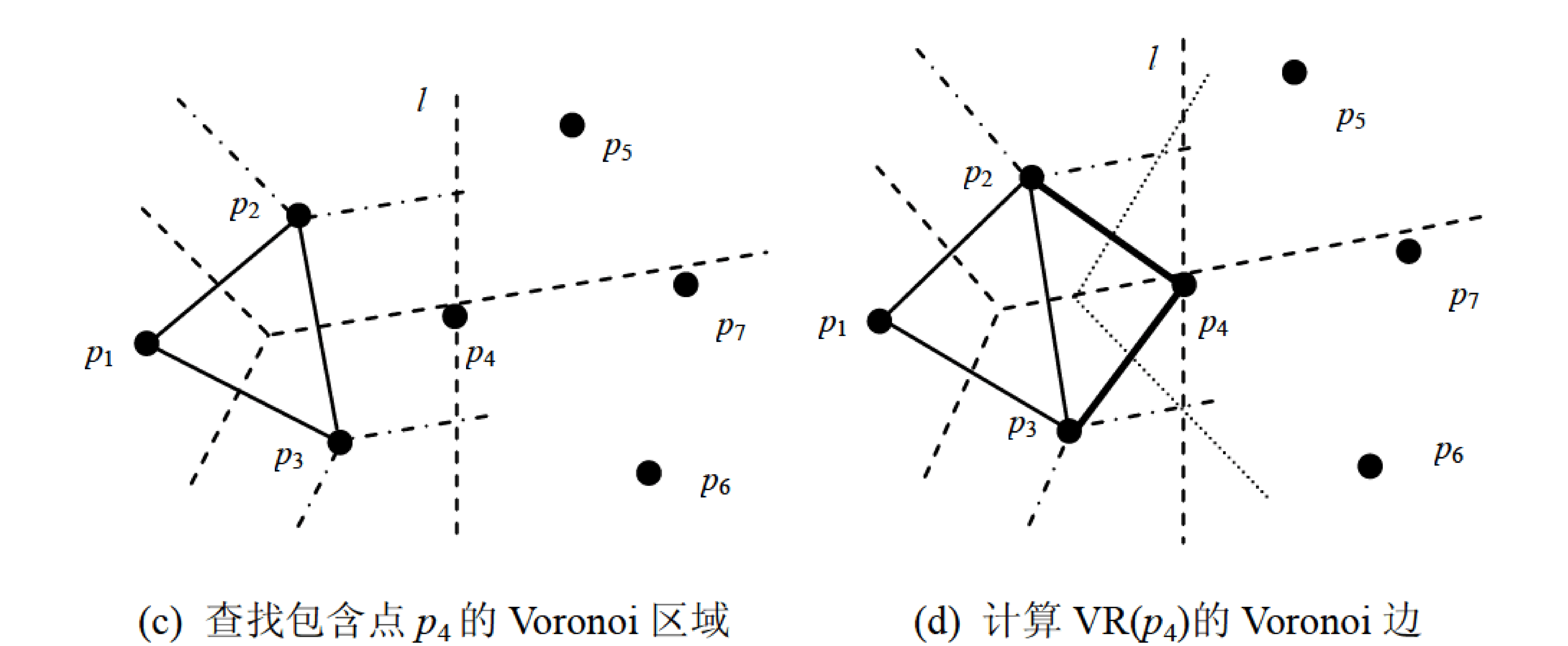
图 2.12 给出了一个计算平面点集的 Voronoi 图的例子。排序后的序列为：p_1，p_2，…，p_7（如图 2.12（a）所示）。图 2.12（b）给出了点 p_1，p_2，

p_3 的 Voronoi 图和 Delaunay 三角网格，其 RCHC 为 p_3 p_2。图 2.12（c）~图 2.12（e）给出了计算点 p_1，p_2，p_3，p_4 的 Voronoi 图的过程，以及最后得到的 RCHC $p_3p_4p_2$。图 2.12（f）~图 2.12（g）给出了计算点 p_1，p_2，p_3，p_4，p_5 的 Voronoi 图的过程，以及最后得到的 RCHC $p_3p_4p_5$。图 2.12（h）给出了计算点 p_1，p_2，p_3，p_4，p_5，p_6 的 Voronoi 图的过程，以及最后得到的 RCHC p_6p_5。图 2.12（i）~图 2.12（j）给出了计算得到的点 p_1，p_2，…，p_7 的 Voronoi 图。



(a) 排序后的平面点集

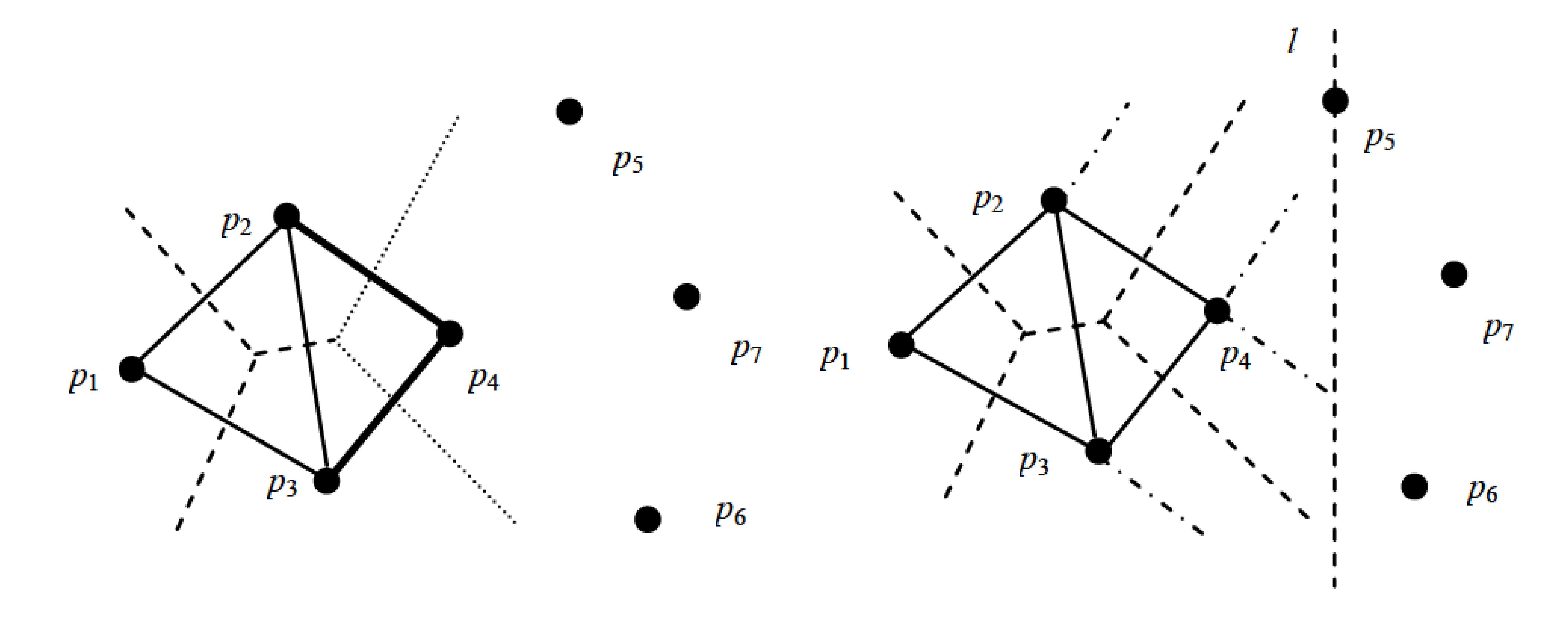
(b) 点 p_1，p_2，p_3 的 Voronoi 图和 Delaunay 三角剖分



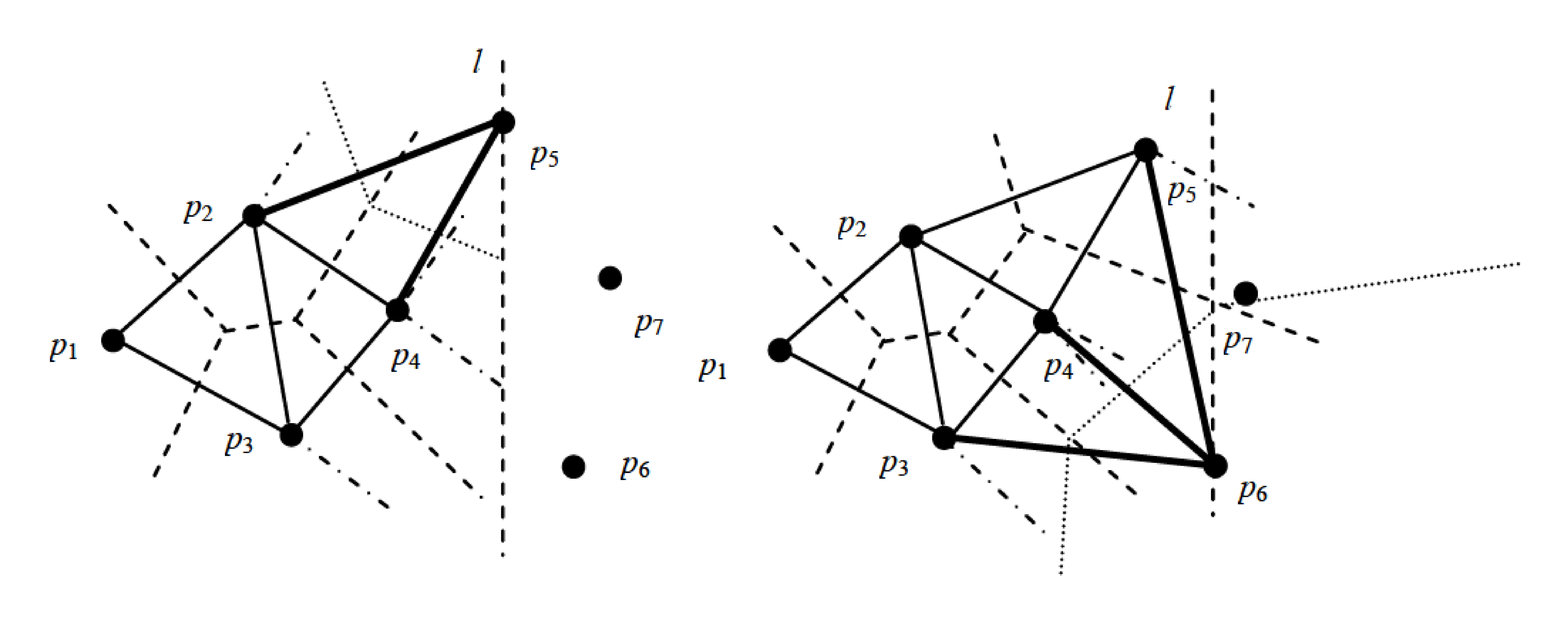
(c) 查找包含点 p_4 的 Voronoi 区域

(d) 计算 VR(p_4)的 Voronoi 边

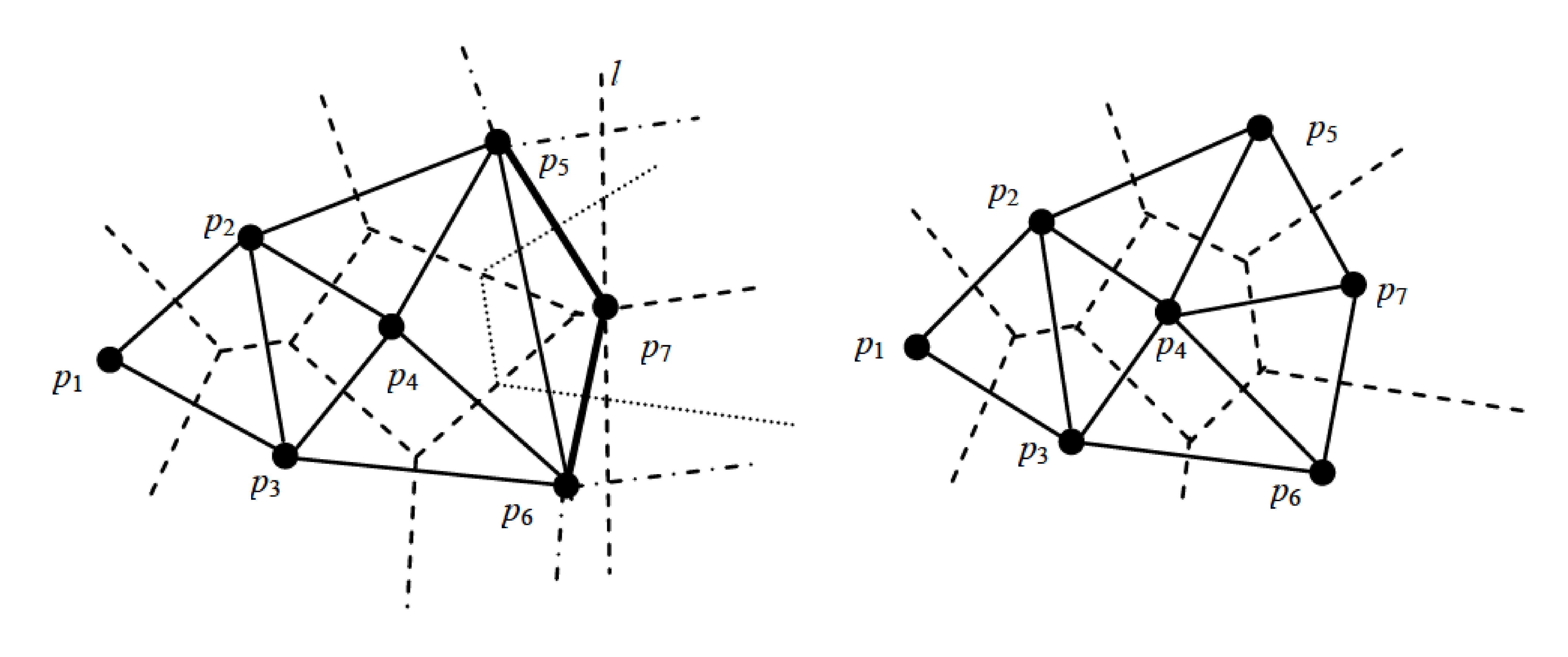
图 2.12　算法过程示意图



(e) 点 p_1，p_2，p_3，p_4 的 Voronoi 图和 Delaunay 三角剖分　　(f) 查找包含点 p_5 的 Voronoi 区域



(g) 计算 VR(p_5)的 Voronoi 边　　(h) 计算 VR(p_6)的 Voronoi 边



(i) 计算 VR(p_7)的 Voronoi 边　　(j) 整个点集的 Voronoi 图

图 2.12　（续）

2.2.4　基于 GPU 生成 Voronoi 图

由于GPU的飞速发展，GPU通用计算（General Purpose GPU）也成为一个热门的领域，出现了很多基于GPU的Voronoi图算法。如Hoff 在1999年提出一种基于GPU计算二维Voronoi图的算法，以每个站点为顶点画一个圆锥体，这些圆锥体正投影到平面上即得到Voronoi图。这个算法线性依赖于所有圆锥体的三角面片数目[Hoff1999]。Denny在2003年提出上述算法的改进，将每个站点的圆锥体替换为一个深度纹理，从而得到了更好的质量和更快的速度[Denny2003]。

这里，我们主要介绍JFA（Jump Flooding Algorithm）算法[Rong2006]，该算法的主要思路是：首先，将给定区域进行像素离散化，表示为一张二维纹理，其中每个像素存储了它的坐标以及最近站点的ID，每个纹理单元都有 r、g、b、a 四个通道，我们用其 r、g 通道记录站点的位置信息，如果当前纹理单元中没有站点，则其 r、g 通道可记为（−1，−1）；然后，算法将每个站点映射到相应的像素中，并将站点信息扩散传递到其周围的像素。扩散过程在GPU上并行执行，因此算法时间不依赖于站点个数，而只与输出图像的分辨率有关。

这个过程是通过 Flooding的思想来实现的。在初始化中，一般将步长 k 设置为2的指数倍，如 $2^{\lceil \log n \rceil}$，例如给定一个站点(x, y)，在第一次Flooding的时候，在顶点程序中计算它的相邻单元（$x+i$，$y+j$），其中，$i,j \in \{-k,0,k\}$，在片段程序中相邻单元计算与传来站点的距离，并将此距离与当前它记录的最短距离相比，取最小值，更新最近站点的信息。依此类推，每次 k 值减半，直到 $k=1$。图 2.13 示意了JFA算法如何从左下角的一个站点开始，通过三次迭代填充一张8×8的纹理。

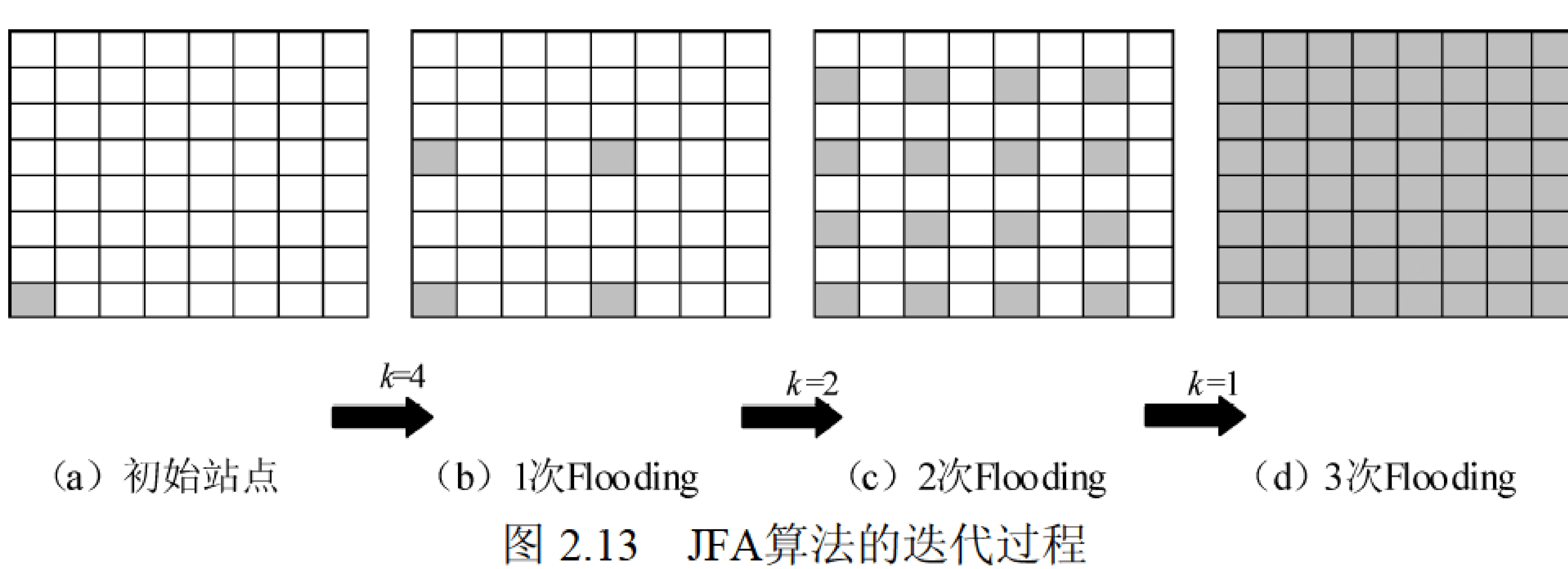


图 2.13　JFA算法的迭代过程

JFA 算法不但适于平面点集的 Voronoi 图计算，也适于一般单元如线段、圆弧等[Yuan2011]，并可用于重心 Voronoi 图的计算[Rong2011]。重心 Voronoi 图的相关介绍参见第 5 章。

2.2.5　基于网格生长的 Delaunay 三角剖分

Fang 和 Piegl 基于网格实现了离散点集的 Delaunay 三角剖分，将 Delaunay 顶点的搜索限制在局部范围内，速度较快；并对随机点集测试表明，时间复杂度呈近似线性[Fang1993]。

该算法的大致步骤如下。

（1）构造均匀网格。

（2）将各站点置入相应的网格单元。

（3）计算初始站点和初始边，保证该初始边为一条 Delaunay 边。

（4）基于均匀网格，根据最大空圆特性快速查找当前 Delaunay 边的 Delaunay 顶点。

（5）选择新的当前 Delaunay 边，重复上面的步骤（4），直到找出所有的 Delaunay 三角形为止。

下面主要介绍步骤（1）～步骤（4）如何实现。

1. 构造网格

对于给定的平面离散站点集合 $P=\{p_1, p_2, \dots, p_n\}$，其中站点 $p=(x_i, y_i)$，算法把站点所在的平面区域划分成均匀网格，使网格单元数和站点数大致相同。

覆盖 P 的均匀网格的构造过程如下。

（1）首先计算 P 的包围盒。

（2）利用下式计算网格单元的大小：

$$size = \sqrt{\frac{(X_{\max} - X_{\min})(Y_{\max} - Y_{\min})}{n}}$$

其中，$X_{\max}$、$X_{\min}$、$Y_{\max}$、$Y_{\min}$ 分别是包围盒的 X、Y 的最大最小值。

（3）然后根据确定的网格单元大小将包围盒均匀分割，得到覆盖 P 的一个均匀网格，其在 x、y 方向上网格单元数为：

$$X_{res} = \mathrm{int}\left(\frac{X_{\max} - X_{\min}}{size}\right) + 1，\quad Y_{res} = \mathrm{int}\left(\frac{Y_{\max} - Y_{\min}}{size}\right) + 1$$

2. 将各站点置入相应的网格单元

根据站点的坐标值，将其放入相应的网格单元中。

为了把每一个站点放入一个且只一个网格单元中，对每个站点 $p_i = (x_i, y_i)$ 执行下列操作。

（1）计算 $cell_i = \mathrm{int}\left(\frac{X_i - X_{\min}}{size}\right)$ 和 $cell_j = \mathrm{int}\left(\frac{Y_i - Y_{\min}}{size}\right)$。

（2）如果单元($cell_i$，$cell_j$)是空的，则把站点(x_i，y_i)放入。

（3）如果单元($cell_i$，$cell_j$)中已经有站点，则检查是否是重合的站点。

a）如果当前站点与单元中已有的站点重合，则忽略该站点。

b）否则，把该站点放入单元中。

这样，每一个数据点属于且只属于一个网格单元，并且自动消除了重合点。

3. 计算初始站点和初始边

在靠近中间的网格单元中找到一个初始站点，然后找一个距离该初始站点最近的站点，并形成一条有向边，称为初始边，其方向为从初始站点到其最近站点。根据性质 2.2 知，这条边为一条 Delaunay 边。

（1）找初始站点

尽管初始点可以从任何站点开始，但为了提高效率，算法选择位于网格中心附近的站点作为初始点。

令 $cell_i = \lfloor X_{res}/2 \rfloor$； $cell_j = \lfloor Y_{res}/2 \rfloor$。如果单元$(cell_i$， $cell_j)$中存在站点，则选取其中一个作为初始点，否则搜索相邻的单元并选取其中一个作为初始点。

（2）计算初始边

在得到初始点 p_1 后，使用下面的方法寻找其最近点，计算初始边。

① 给 d_{min} 赋一个比较大的初始值，这里令其为包围盒的对角线的长度；

② 如果包含初始站点的单元中含有一个以上的站点，则寻找其中距离 p_1 最近的站点，计算这两点之间的距离 d ，如果 $d < d_{min}$， 则 $d_{min} = d$ ；

③ 按一定的行列顺序搜索相邻的单元。

从 p_1 到靠近 p_1 的网格边画一条垂线；如果垂线的长度小于 d_{min}，则搜索这一行或列；否则，把这个方向（为上、下、左、右之一）标记为无效方向（因为沿这个方向搜索的网格单元中的站点到 p_1 的距离大于等于 d_{min}，根据性质 2.2，该站点和 p_1 之间的边不是 Delaunay 边）。如果发现了站点，计算其到 p_1 的距离 d ，如果 $d < d_{min}$， 则 $d_{min} = d$ ；当所有方向都被标记为无效方向时，停止搜索（因为所有在未搜索行列中的站点到 p_1 的距离都大于 d_{min} ）。由于点集中站点数 $n > 2$ ，所以该方法最后可以找到初始边。根据性质 2.2，初始边是 Delaunay 边。

4. 构造 Delaunay 三角形

设当前处理的边（有向边）为 $p_i p_j$ ，扫描这条边的右边，寻找与这条边的两个端点所形成的角中角度最大的站点。然后扫描这个站点与当前边所形成三角形的外接圆所覆盖的区域。如果外接圆内存在一个站点，计算这个新

站点和当前边所形成三角形的外接圆，再扫描新外接圆所覆盖的区域，直到新找到的站点与当前边所形成三角形的外接圆所覆盖区域内没有站点为止。根据性质 2.2 知，最后查找得到的站点与这条边所形成的三角形即为 Delaunay 三角形。

具体地，为了寻找第三点 p_k，使三角形△$p_i p_j p_k$ 为 Delaunay 三角形，即满足最大空圆特性，算法执行下列操作。

（1）在 $p_i p_j$ 的右边搜索，寻找单元(i_1，j_1)和(i_2，j_2)，这两个单元或是端点 p_i、p_j 所在的单元或是它们的直接相邻的单元。可以通过 $p_i p_j$ 与 p_i 所在单元的底线相交计算 i_1，交点所在单元的列下标为 j_1。类似地，用 p_j 所在单元的右边网格线计算 j_2，交点所在单元的行下标为 i_2（如图 2.14 所示）。

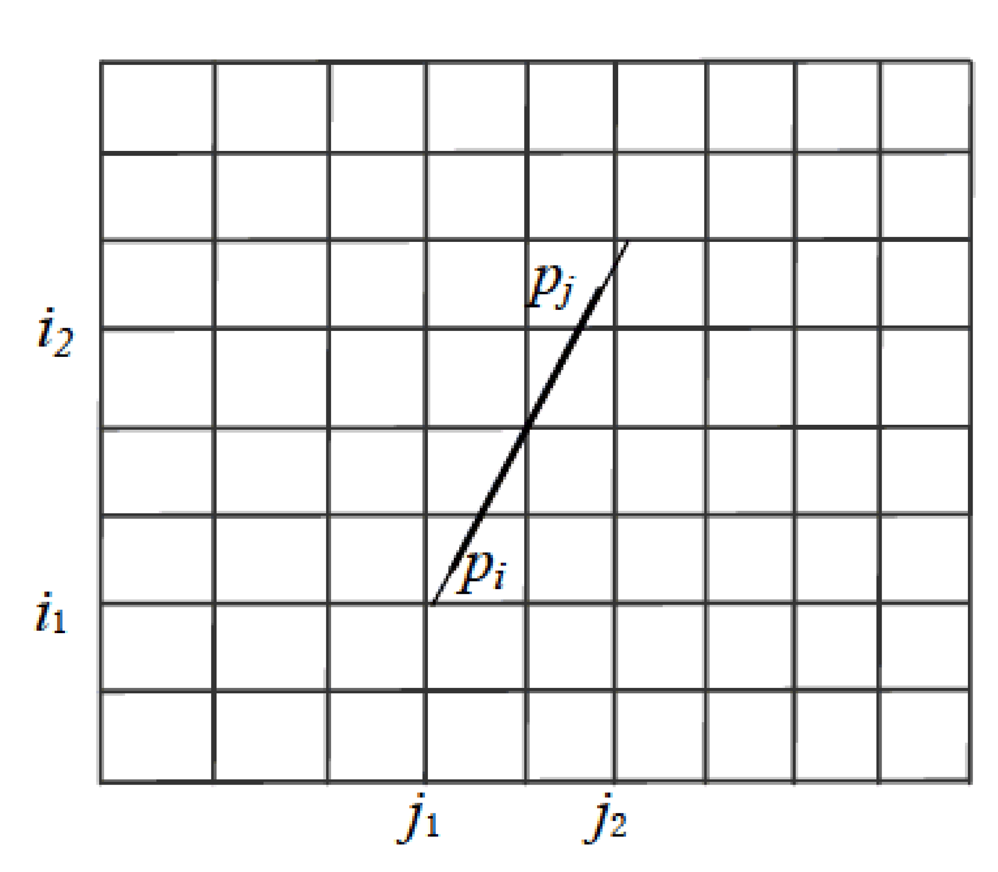


图 2.14　寻找 Delaunay 顶点

（2）这样，形成了由单元覆盖的三角形区域。该单元三角形的顶点是(i_1，j_1)、(i_1，j_2)、(i_2，j_2)，边界是水平方向、垂直方向和对角线方向排列的单元。在对角线方向选择与直线 $p_i p_j$ 相交的单元。

（3）检查区域内的每一个单元：从(i_1，j_2)开始搜索每一列，往左移动直到(i_1，j_1)。这样，首先搜索的单元最有可能包含所要求的站点，使生成的三角形在第三站点的角度最大，尽可能接近于等角三角形。

（4）如果步骤（3）发现了多个站点，则选择给出最大角的站点；如果没有发现站点，则继续从(i_1-k，j_1-k)到(i_1-k，j_2+k)搜索行，从(i_1-k，j_2+k)到(i_2+k，j_2+k)搜索列($k=1$，2，…)，直到发现站点为止。

（5）通过这三个站点作外接圆，在获得第一个圆后，继续在圆所覆盖的网格单元中搜索，做同样的处理，即选择具有最大角的站点。计算该站点与当前边所形成的三角形的外接圆，如果在该外接圆所覆盖的网格单元中都搜索完以后，没有发现其他站点，则说明该站点是要找的站点，形成的三角形为 Delaunay 三角形，这时停止搜索。

动态修改圆是这个算法的关键部分，因为搜索局限于靠近边 $p_i p_j$ 的部分进行，一般只用几步就可找到点。如果边 $p_i p_j$ 是水平的或垂直的，可以使用简单的包围盒代替三角形。

5. 算法正确性证明

下面证明这个算法可产生有效的 Delaunay 三角形，即对于每一个计算得到的三角形，其外接圆内不含任何其他站点。由于算法只是搜索每条边的右边，这样可以保证该边的右边在外接圆内没有其他站点（既然算法使用具有最大角的站点，在圆内该边的右边没有站点）。但是这里还需要证明在外接圆内该边的左边也没有其他站点。

图 2.15 中，A 为起始点，B 是距离 A 最近的站点，因此对于任何与边 AB 构成三角形的其他点 C，$|AB|\leqslant|AC|$，故 $\angle ACB\leqslant\angle ABC$，$\angle ACB\leqslant 90°$，不含 C 的弧 AB 小于半圆弧。如果在△ABC 的外接圆内有一站点 D 且位于 AB 的左边，则$|AD|\leqslant|AB|$，这与 B 是距离 A 最近的站点矛盾。因此在△ABC 的外接圆内不含任何其他站点。

找到第一个 Delaunay 三角形 ABC 以后，以 AC 为起始边继续在 AC 的右边寻找第三站点 D，根据算法中使用的搜索原则，可以保证在△ACD 的外接圆内 AC 的右边没有其他站点，但是同样需要证明在△ACD 的外接圆内 AC 的左边也没有其他站点。

如图 2.16 所示，因为 D 在圆 ABC 的外边，所以圆 ACD 与 AC 的垂直平分线的交点 Q 一定位于圆 ABC 的外面且与点 P（圆 ABC 与 AC 的垂直平分线的交点）在 AC 的同一侧，否则点 D 或者在圆 ABC 内或者在 AC 的左边，

都会出现矛盾。因此弧 *AQC* 的补弧全部包含在圆 *ABC* 内，且在 *AC* 的左边。由于圆 *ABC* 内不含任何数据点，因此在 *ACD* 的外接圆内 *AC* 的左边也没有数据点。

依此类推，可以确定该算法生成的三角形是 Delaunay 三角形。

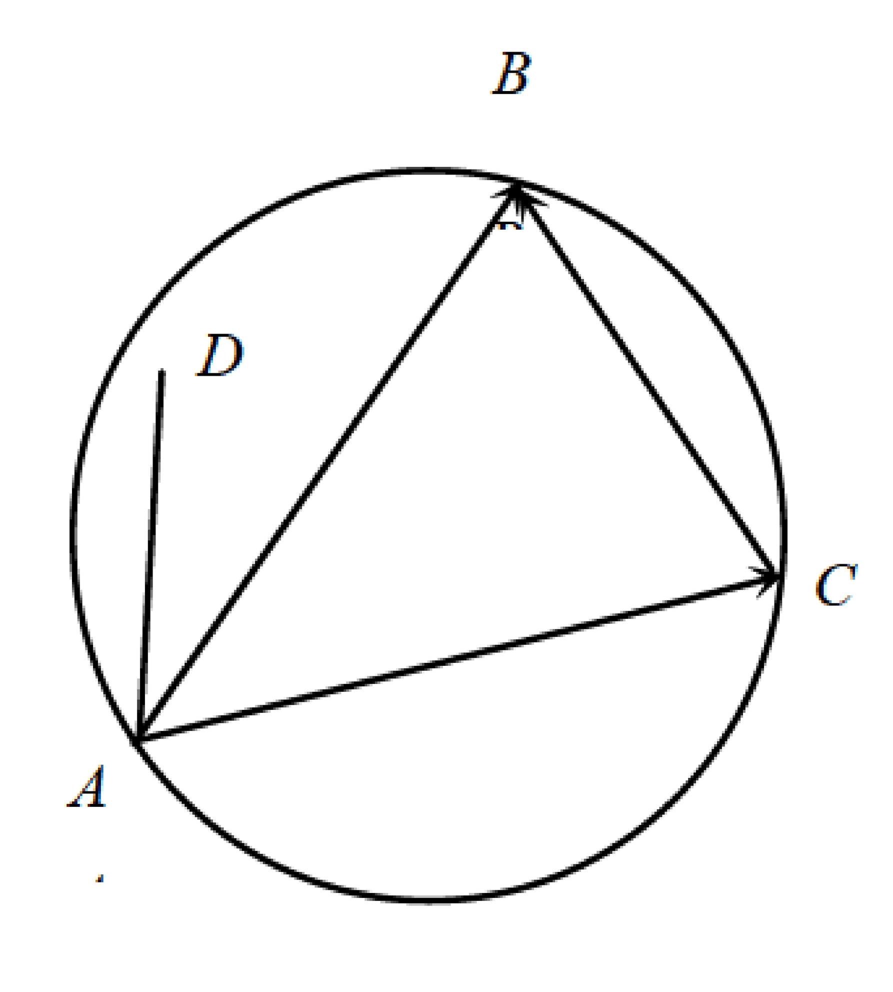


图 2.15　初始三角形

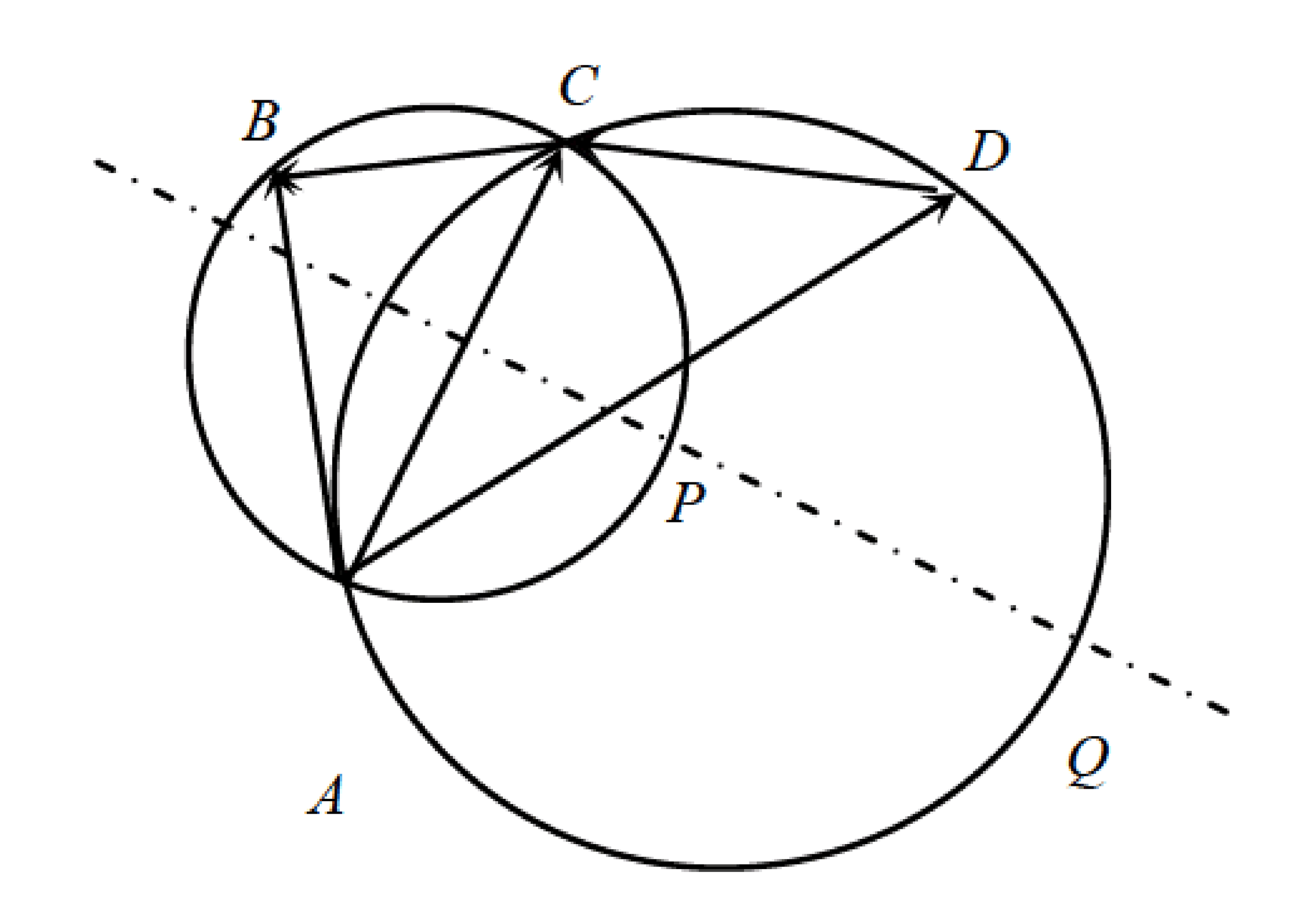


图 2.16　构造第二个三角形

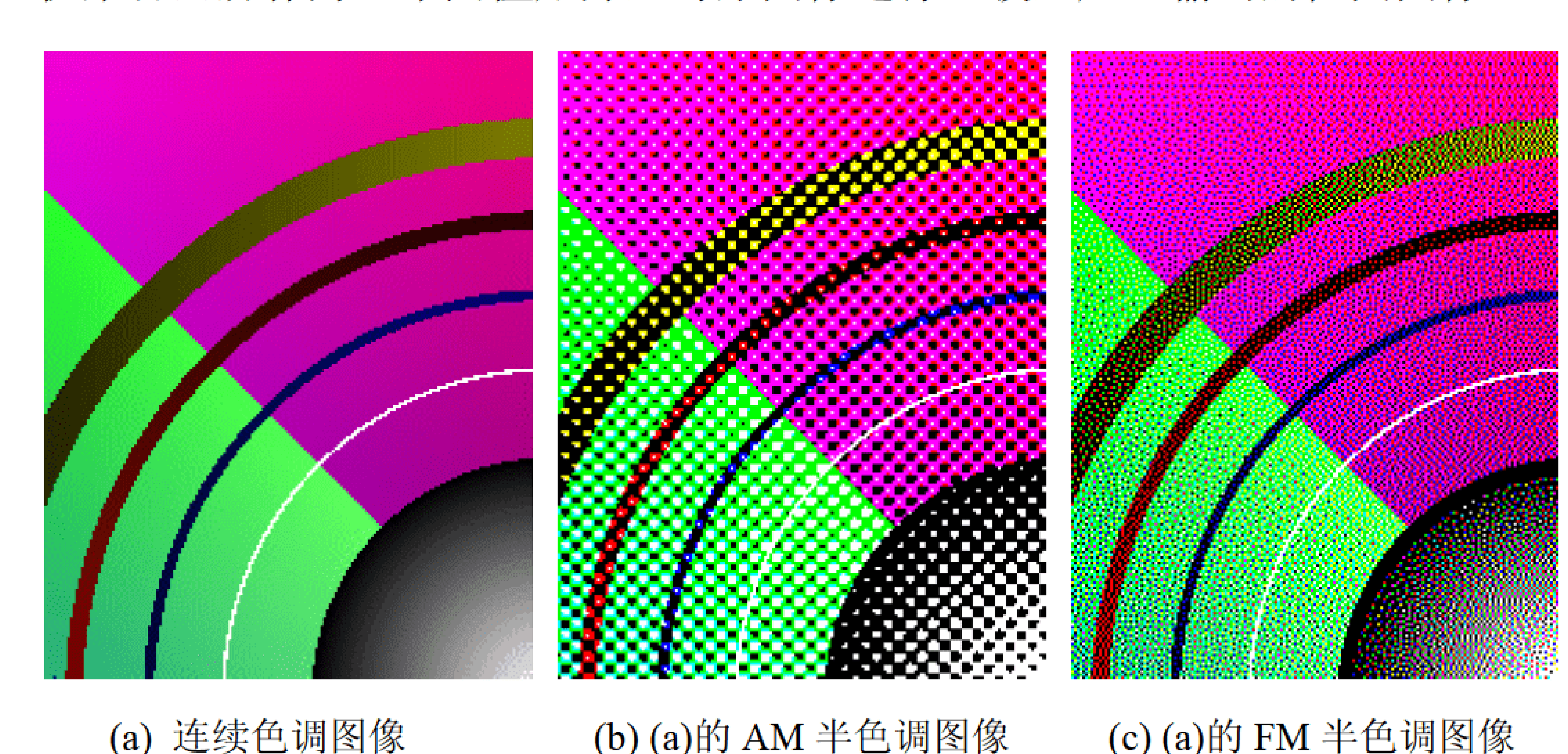
2.3　应 用 实 例

2.3.1　半色调图像生成

一般地，我们在屏幕上编辑的图像常常是连续色调的（如图2.17（a）所示）。而对于图像的印制来说，一个墨点在印刷品上或有或无，并且墨点总是相同的单一的颜色。因此，如何使用单一的颜色（黑色）模仿出百余种灰度等级，或只用几种颜色（如印刷中的四色青、品、黄、黑）模仿出几百万种颜色，成为图像印制输出中的一个关键问题。

将连续色调的图像转化为二值图像的技术被称为半色调技术。照片、图画等连续色调图像在进行大批量印刷之前必须要离散为半色调形式，工业上称之为挂网。数字半色调技术是由传统的半色调技术发展而来的。其中误差扩散是常用的一种半色调技术。传统的误差扩散是把一个像素的量化误差扩散给相邻的像素，使像素的量化误差在另外的像素中得到补偿。误差扩散技术存在的一个主要问题是固定的误差传播模式会在灰度平坦的地方和平滑的

灰度梯度图中生成规则的周期图案。在印染、印刷行业中，考虑到印制所用油墨与染料、印制扩散、印制速度、印制材料等原因，常用数字半色调技术先生成聚合网屏（抖动矩阵），根据聚合网屏中设定的阈值再决定输出的图像值（黑或白）－实际挂网时，是把产生的随机聚合网屏叠加在图像上，把随机聚合网屏作为一个阈值矩阵，与原图像进行比较，产生输出的位图图像。



(a) 连续色调图像　　(b) (a)的 AM 半色调图像　　(c) (a)的 FM 半色调图像

图 2.17　连续色调图像及其半色调图像

数字半色调设备，如激光打印机和激光照排机，它们只能产生固定大小的点，称之为元素点。元素点与半色调网点、图像分辨率所指的点都不相同。元素点（Printer Dots或Points）指输出设备所能描绘的最小点单元，它是最小单位元素，它的大小可以从输出设备的输出分辨率中看出。半色调网点（Halftone Dots）是由元素点组成的，但不一定只有一个元素点。为了模仿出各种不同大小的半色调网点，输出设备将固定大小的元素点放入每个半色调网格中。半色调网格叠加在图像上，产生半色调网点。按照网点的分布规律，数字半色调也可以分成三大类：调幅挂网、调频挂网、调幅调频混合挂网。调幅网屏是在一个规则的网格中均匀地放置各种尺寸的网点（一簇点），网点之间的距离保持固定不变，用变化网点大小的方法来模拟不同的色调，较大的网点处产生暗色调，较小的网点处产生亮色调。如图2.17（b）给出了这种技术产生的图2.17（a）中图像的半色调图像。这种方法的缺点是规则的网屏

图案在印刷中会产生明显的干扰图案，不同角度和频率的网屏叠加时会产生龟纹（Moirés）。调频网屏则是网点的大小（幅度）不变，变化的只是网点出现的频率，以网点的疏密来表现颜色的深浅，颜色深则网点密，颜色浅则网点疏，网点密度高的区域表现暗色调，而网点密度低的区域表现亮色调。如图2.17（c）给出了这种技术产生的图2.17（a）中图像的半色调图像。调频网屏可以有效地避免网屏叠加时干扰图案和云纹现象的发生。较之调幅网屏，调频网屏使用的网点要小得多，可表现更加丰富的颜色及灰度过渡层次以及图像细节。调幅调频混合网屏则是将大小不同的网点放置在随机的位置，以取得更好的效果。调频网屏和调幅调频混合网屏都称为随机网屏（Stochastic Screening）。但是随机网屏需要用复杂的算法来决定网点的位置，既要有随机性，又不能造成视觉上的颗粒化。虽然这种技术有着很大的市场潜力，但是由于算法占用内存多、计算量大，因此要求输出设备有很高的计算性能。

本节介绍一种基于Delaunay三角剖分的随机聚合网屏的生成算法[Tu2000，Pan2001]。算法在随机聚合网屏的生成过程中，根据随机网屏中网点的分布特点，对放置的平面点集进行快速Delaunay三角剖分，提高了网屏生成的速度。

算法包含以下三个主要步骤。

（1）在大抖动矩阵内随机放置网点，获得网点中心。

（2）对网点中心进行 Delaunay 三角剖分。

（3）填充网点，产生随机聚合网屏。

1. 随机放置网点

为了得到抖动矩阵内所有网点中心位置的空间分布，首先把抖动矩阵的所有元素标记为自由元素，然后沿着一条能遍历抖动矩阵所有元素的访问路径访问每一个元素。当遇到一个自由元素时，以该元素为中心放置一个半径为 r 的圆盘（ r 的大小应根据印刷过程和印刷材料来决定，一般有一个变化

范围，由用户选择），并把这个圆盘覆盖的所有元素标记为占用元素。由于这条访问路径能遍历所有的矩阵元素，最后整个矩阵就由部分重叠的圆盘所覆盖。所有圆盘的中心构成了网点的中心。

在抖动矩阵内放置网点时，一般应该遵循下列要求。

（1）应避免出现低频成分，即在绘制的网屏图案中最具代表性的频率应当和网点周期相对应。

（2）在绘制的网屏图案中频率应当是等方向性的，即不应当在某个方向有较强的网屏频率。

（3）由于产生的抖动矩阵可能只覆盖图像的一部分，当沿水平和垂直方向重复时，应当不间断地覆盖整个图像，因此抖动矩阵在边界处应连续，即矩阵应当在水平和垂直方向能够环绕起来。

为了满足上述网点分布（1）和（2）的要求，采用了一种称为随机空间填充曲线的访问路径。首先构造一个包含抖动矩阵所有元素坐标值(x，y)的元素表，其中元素按抖动矩阵的行列顺序排序；然后随机选取有序表中的两个元素进行交换，经过大量的随机交换以后，元素的排列次序完全随机，这个有序元素表可以看作一条随机空间填充曲线。在抖动矩阵中相邻的元素在随机空间填充曲线中一般不会相邻，访问元素的顺序不会对称，产生的网点中心分布是等方向性的，而且，低频成分比在旋转 Hilbert 空间填充曲线中更不显著。

为了满足上述网点分布（3）的要求，为了使抖动矩阵在边界处保持连续，在沿随机空间填充曲线放置圆盘时，可以采用补点的方法。

（1）如果一个圆盘全部落在抖动矩阵内部，则把所覆盖的抖动矩阵元素全部标记为占用元素。

（2）如果一个圆盘部分覆盖抖动矩阵的元素，则把所覆盖的抖动矩阵元素标记为占用元素；把圆盘落在抖动矩阵上面的部分补在抖动矩阵的下边，并且

把这一部分覆盖的抖动矩阵元素也标记为占用元素。用同样的方法处理圆盘落在抖动矩阵下面、左边、右边、左上角、右上角、左下角、右下角的部分。

如图 2.18 所示，左图是没有补点的情况，中间图是在上、下、左、右、左上角补点的示意图，右图表示了右上角补点的情况。经过补点以后，产生的网点中心分布如图 2.19 所示。图 2.20 列出了其相应的 Fourier 频谱图。从其 Fourier 频谱图可以看出，沿随机空间填充曲线得到的网点中心分布是等方向性的，也没有显著的低频成分。

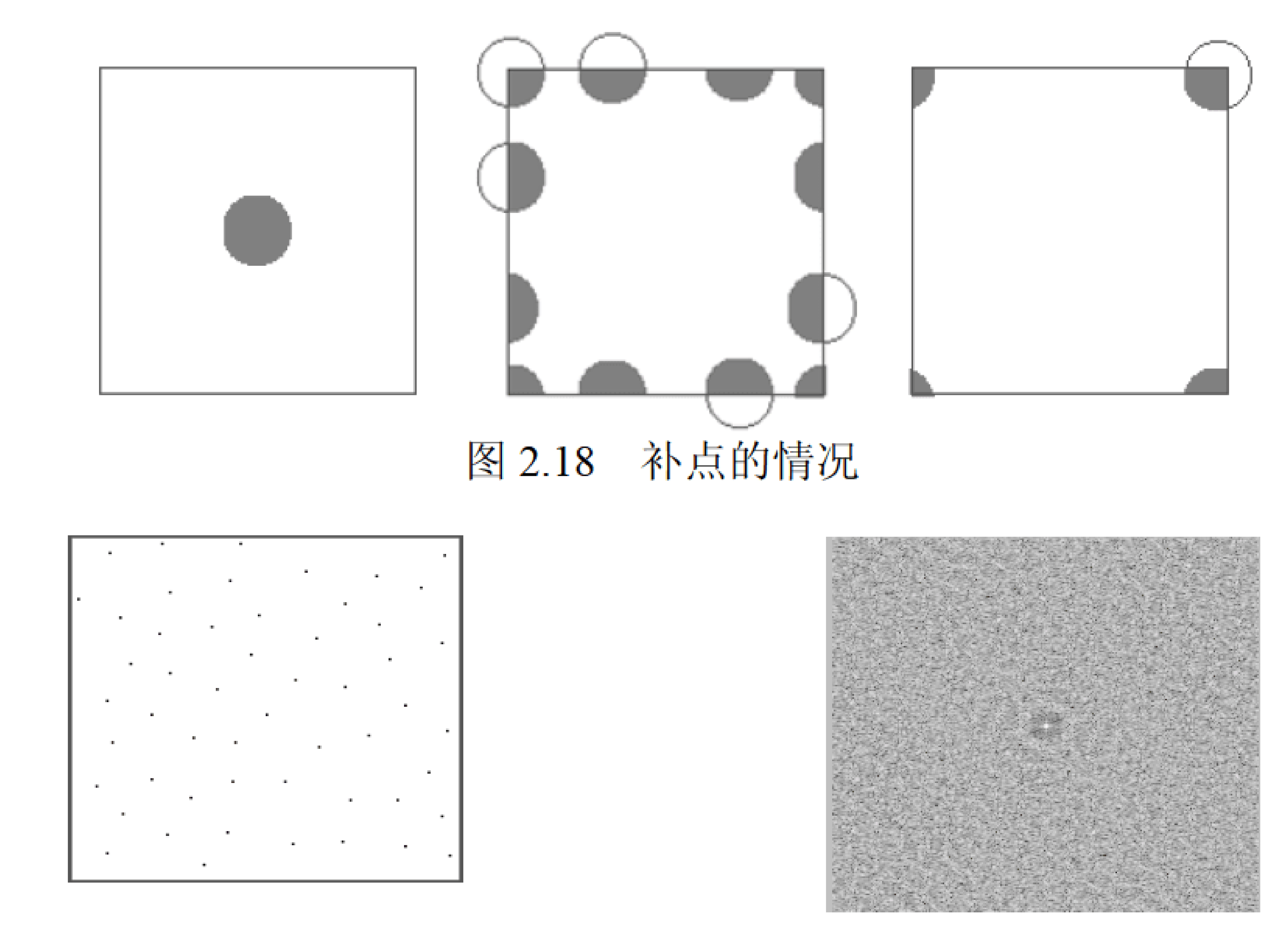


图 2.18　补点的情况

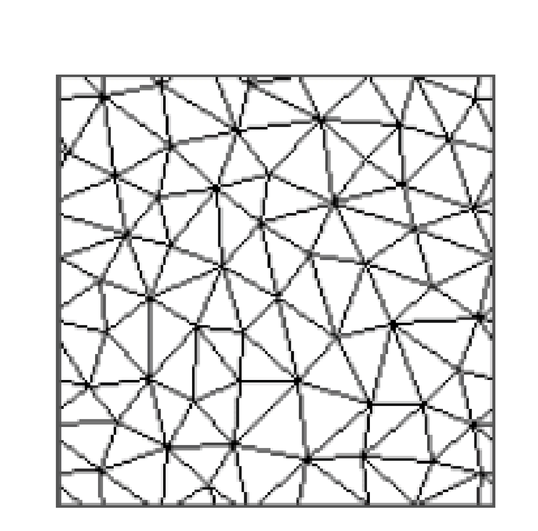
图 2.19　网点中心的分布　　　图 2.20　网点中心的分布的Fourier频谱图

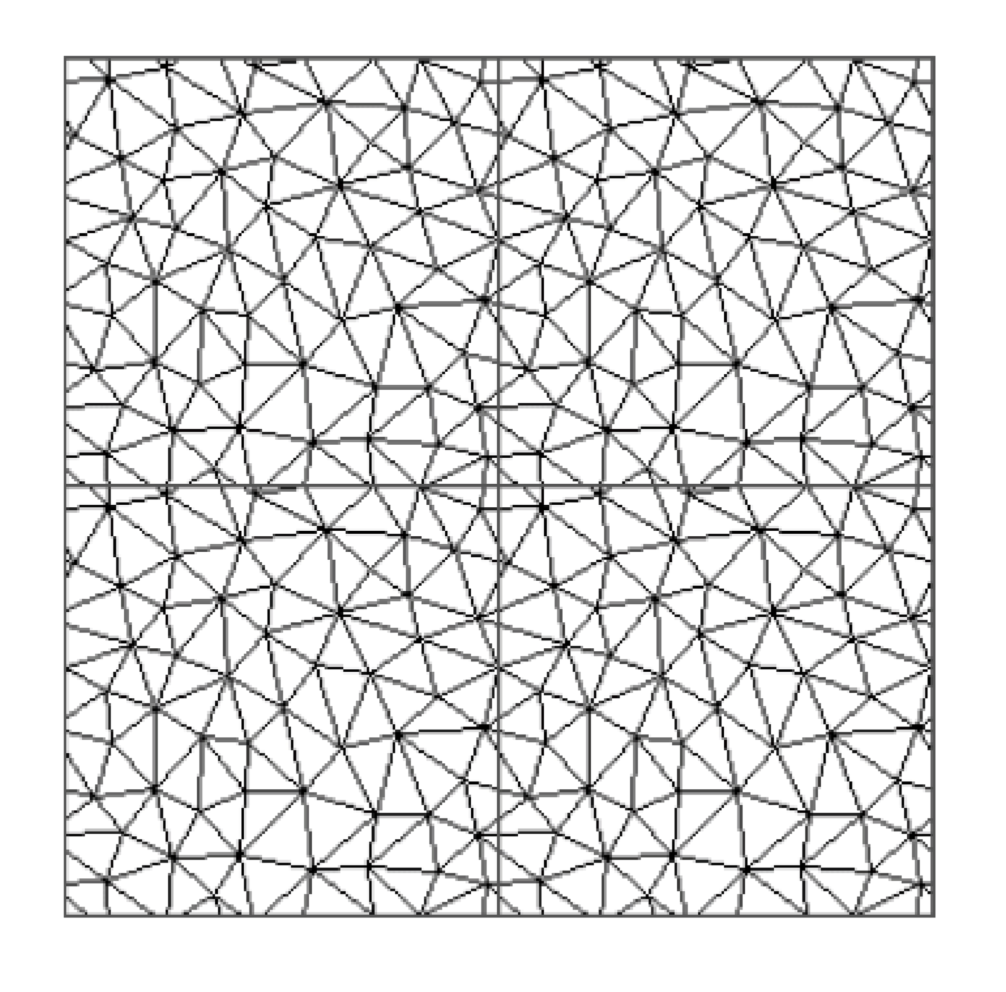
2. 对网点中心集合进行 Delaunay 三角剖分

为了获得每一个网点的最大覆盖区域，对上面产生的网点集实施 Delaunay 三角剖分，然后产生每一个网点的最大覆盖区域。

为了保证抖动矩阵在边界处连续，这里同样采用了补点的方法。把抖动矩阵上面的一部分（这里取抖动矩阵高的四分之一）补在抖动矩阵的下边；把抖动矩阵下面的一部分（这里取抖动矩阵高的四分之一）补在抖动矩阵的上边；把抖动矩阵左面的一部分（这里取抖动矩阵宽的四分之一）补在抖动

矩阵的右边。用同样的方法补充抖动矩阵左边、右边、左上角、右上角、左下角、右下角的部分。如图 2.21 所示为网点中心三角剖分的结果，图 2.22 表示了三角剖分后矩阵的连续性。





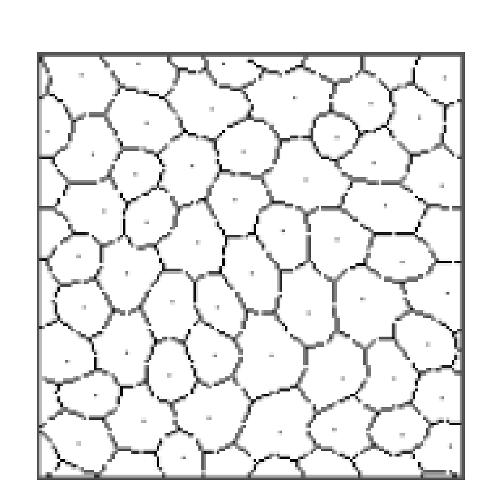


图 2.21 网点中心的三角剖分 图 2.22 三角剖分后矩阵的连续性 图 2.23 网点中心的Voronoi图

3. 生成随机聚合网屏

我们把一个网点的覆盖区域定义为由三角形的重心和各条边中点的连线所围成的区域，即其 Voronoi 区域，如图 2.23 所示。三角形的重心为灰度值最大的点，网点中心为灰度值最小的点。

围绕每一个网点中心，根据要求的灰度级产生等灰度区域，按照灰度值从小到大的顺序、以适当的比例填充三角形内的等灰度区域。因为两个相似三角形的面积之比等于对应边之比的平方，等灰度区域和三角形边的交点的位置可以根据下面的方法计算。

如图 2.24 所示，A、B、C 为相邻的三个网点中心，点 D 是三角形的重心，点 E、F、I 分别是三角形边 AB、AC、BC 的中点。可以证明三角形△DEF 的面积是三角形△AEF 的面积的三分之一，△DEF 内的灰度级是整个网屏灰度级的四分之一；△AEF 内的灰度级是整个网屏灰度级的四分之三。例如，点 D 的灰度级为 255，点 A 为的灰度级为 0，则点 E、F 的灰度级为 192。再在 AE 和 AF 上取点 G 和 H。令 $s_i/s=|AG/AE|*|AH/AF|$，其中 s 表示△AEF 的面积，s_i 表示△AGH 的面积。这里，令线段 GH 平行于线段 EF，在指定 s_i/s 值的情况下，即指定灰度比例的情况下，由上面公式即可得到点 G 的坐标和 H 的坐标。同样，按照上面的方法处理△DEF 以及另外两个四边形。

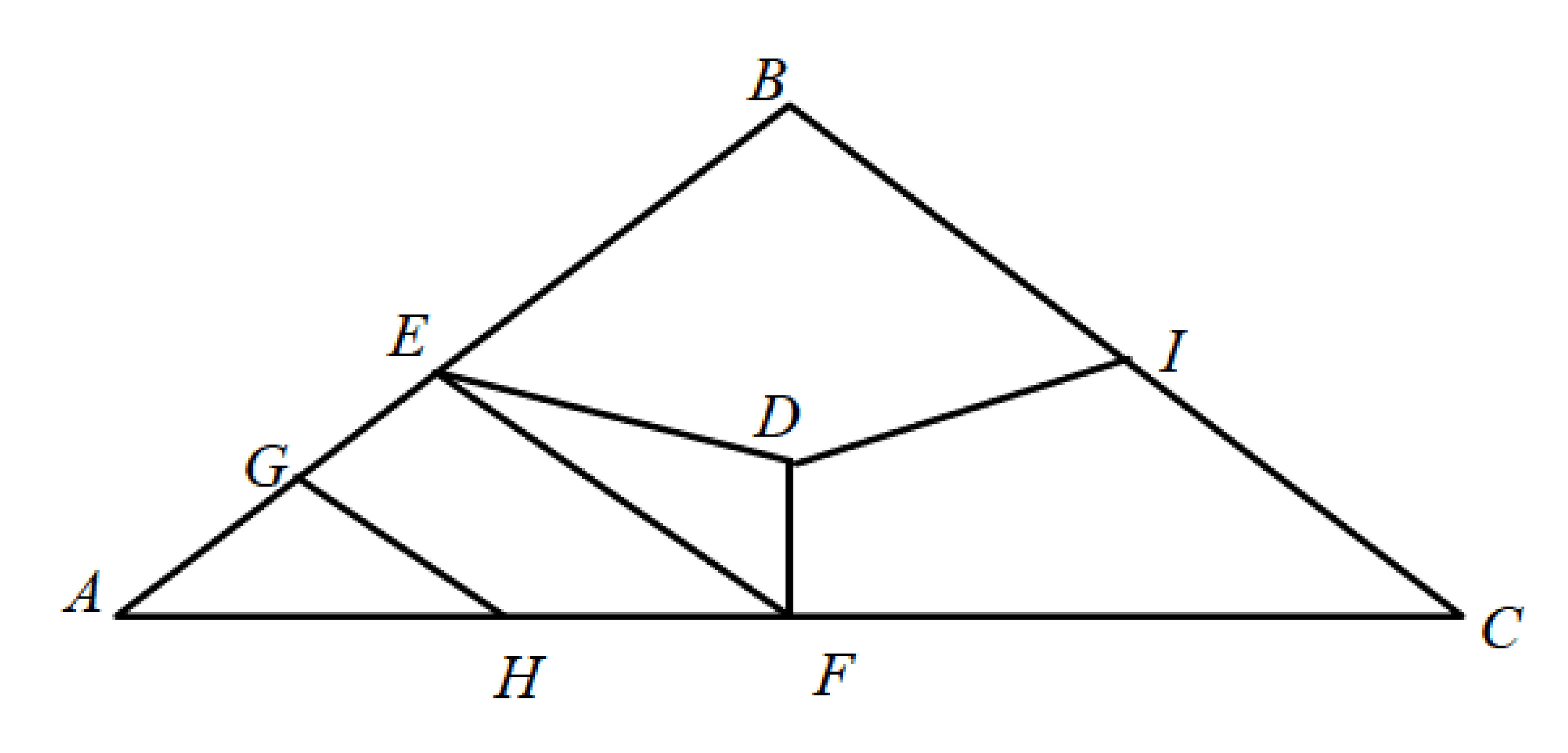


图 2.24　三角形内的等灰度区域

上述方法产生的网点边缘不很平滑，如图 2.25 所示。为了使网点更光滑，可以采用下面的插入附加点的方法。如图2.26所示，把△AEF再分成△AEP和△APF，其中P是边EF的中点。对△AEP做如下处理：取点G和H，有s_i/s=|AG/AE|*|AH/AP|，其中s表示△AEP的面积，s_i表示三角形△AGH的面积，这里取|AG/AE|≠|AH/AP|，即把边AE和AP以不同的比例分割。在指定s_i/s(即灰度比例)的情况下，可由以下公式计算点G的坐标。

令|AG/AE|=λ_2，|AH/AP|=λ_1，$\lambda_2=\lambda*\mu(I)$，$\lambda_1=\lambda/\mu(I)$，则

s_i/s=|AG/AE|*|AH/AP|

$=\lambda_2*\lambda_1$

$=\lambda^2$

这里，$\mu(I)\geqslant 1$，且$\mu(I)$是I的减函数。我们取$\mu(I)=\cos(\pi/2*I/I_s)/k+1$，$I$的取值范围为[0，$I_s$]，$I_s$是△$AGH$中最大的灰度值，$k$控制平滑的程度。经过推导可以得到点$G$的坐标。同理可得到点$H$的坐标。亦可按照上面的方法处理△$DEF$等。

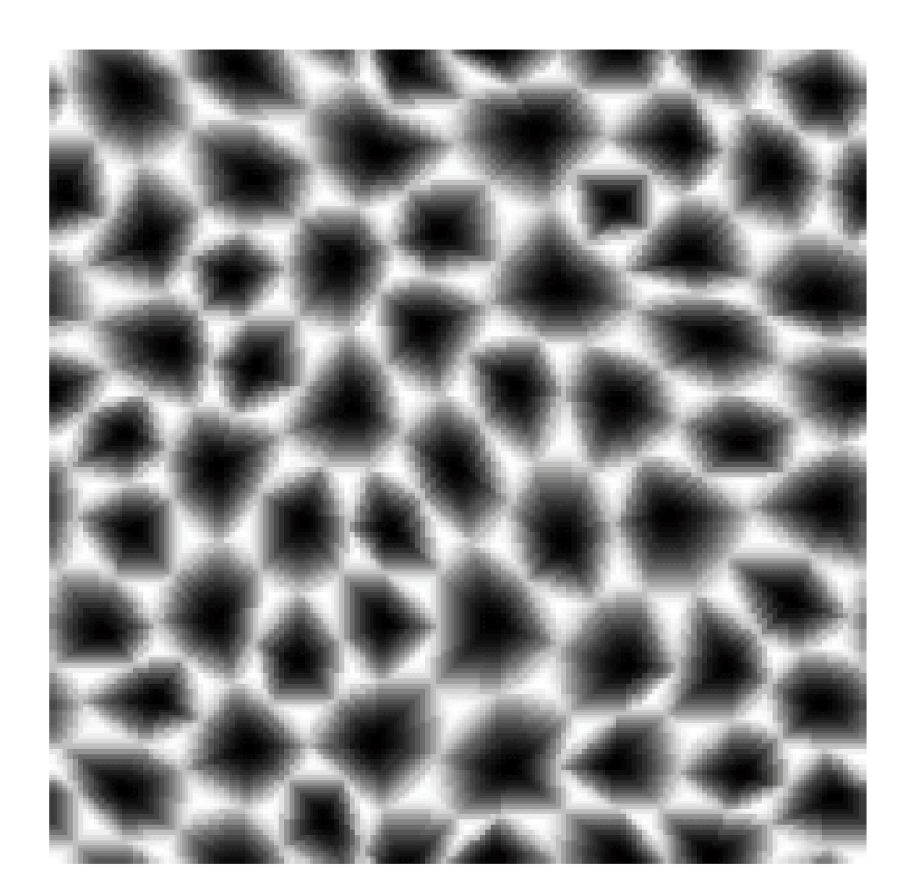


图 2.25　未平滑的网屏

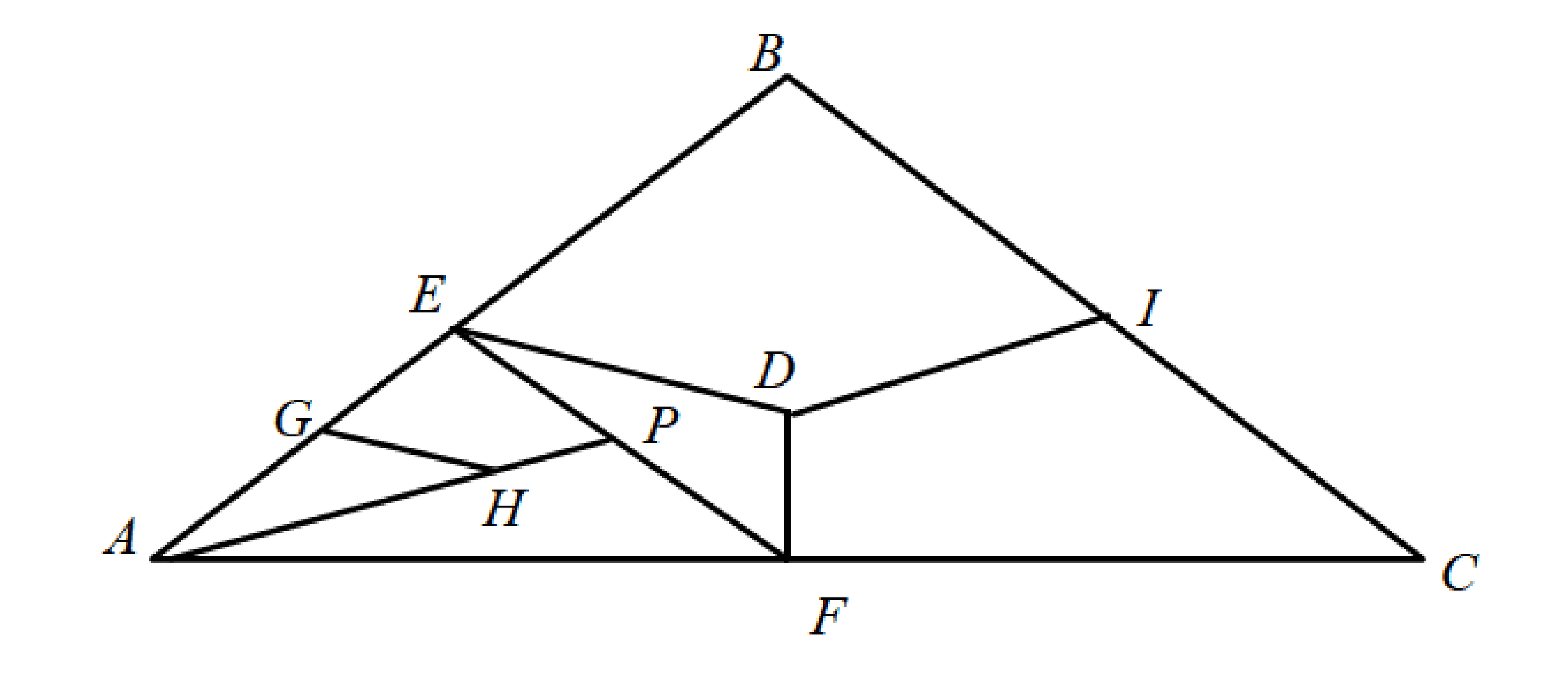
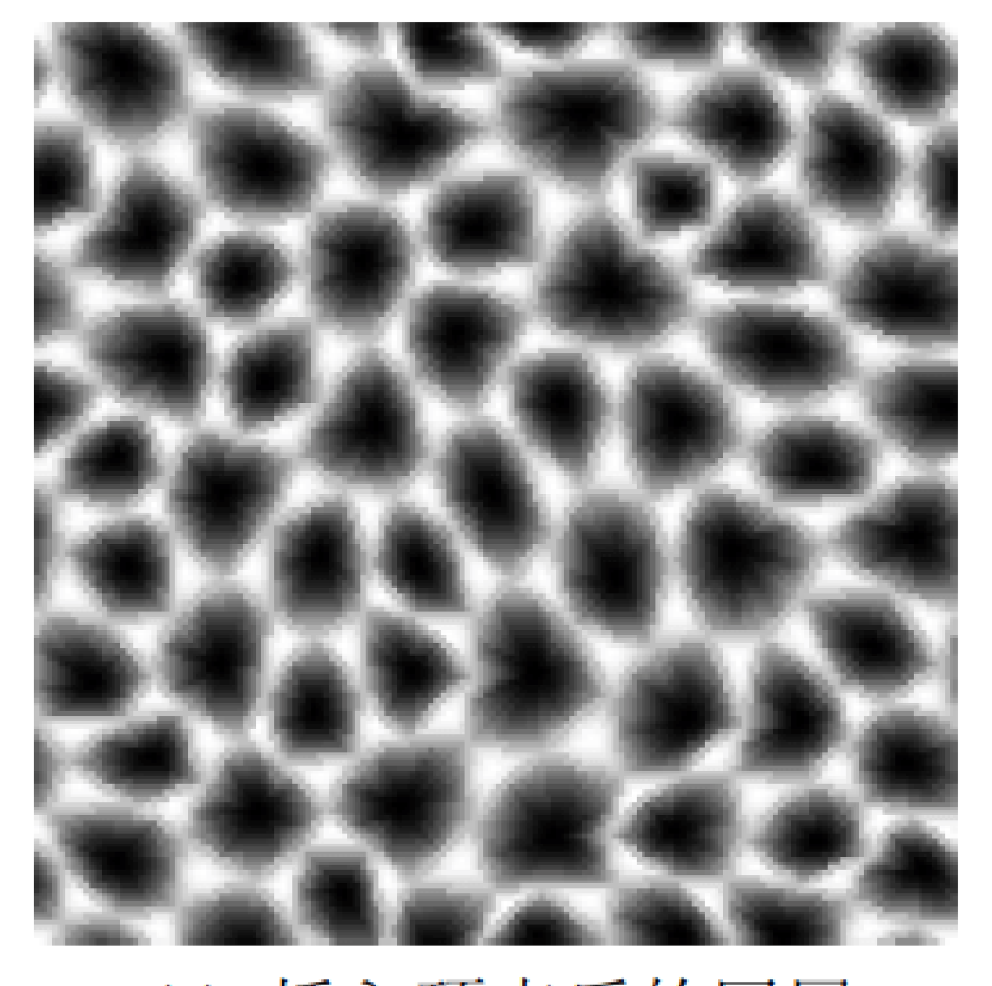
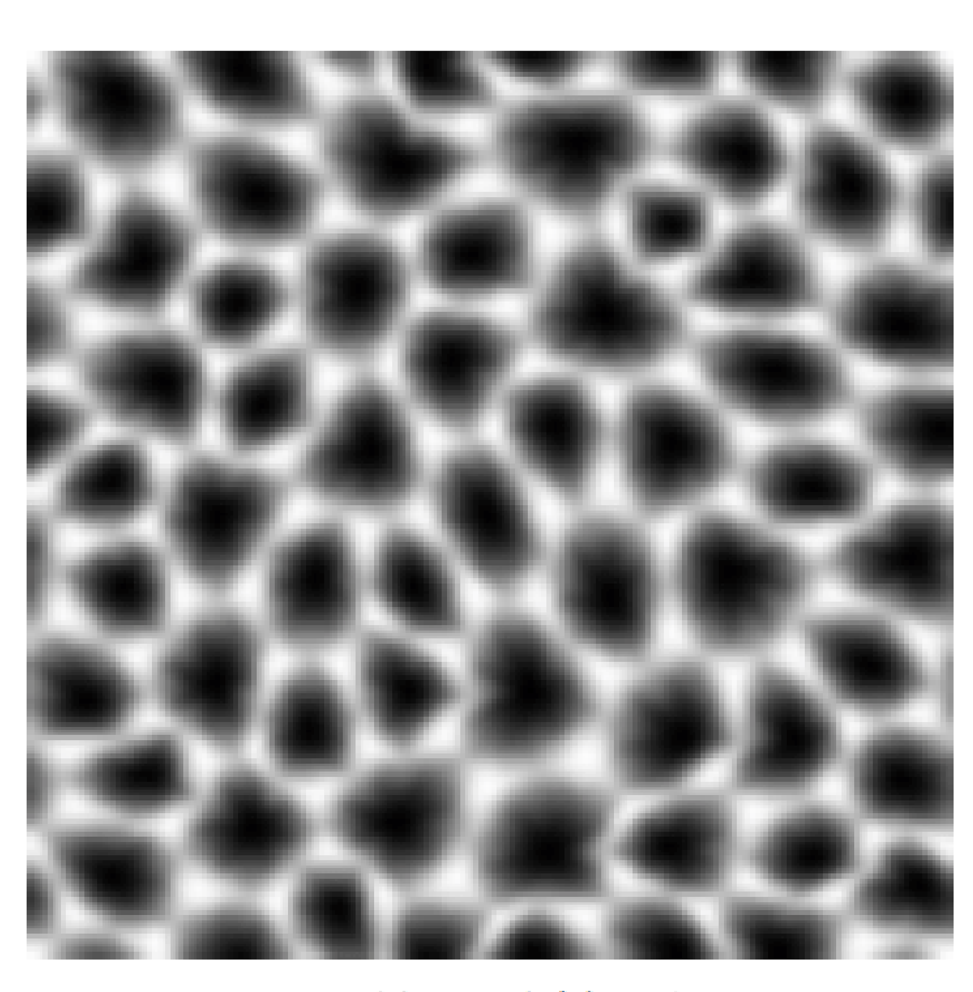


图 2.26　插入附加顶点的方法

为保证灰度值的均匀分布，使生成的网屏具有较平的直方图，需要对生成的网屏利用邻域均值算子进行平滑处理[Kenneth1996]。图2.27（a）是插入顶点后的网屏，图2.27（b）是平滑后的网屏。



(a) 插入顶点后的网屏



(b) 平滑后的网屏

图 2.27　网屏结果

图2.28～图2.30给出了一个例子。图2.28是原图像，图2.29是网屏叠加在图像上的情况，图2.30是在1200dpi下采用半径为8的网屏输出的结果图。



图 2.28　原图像

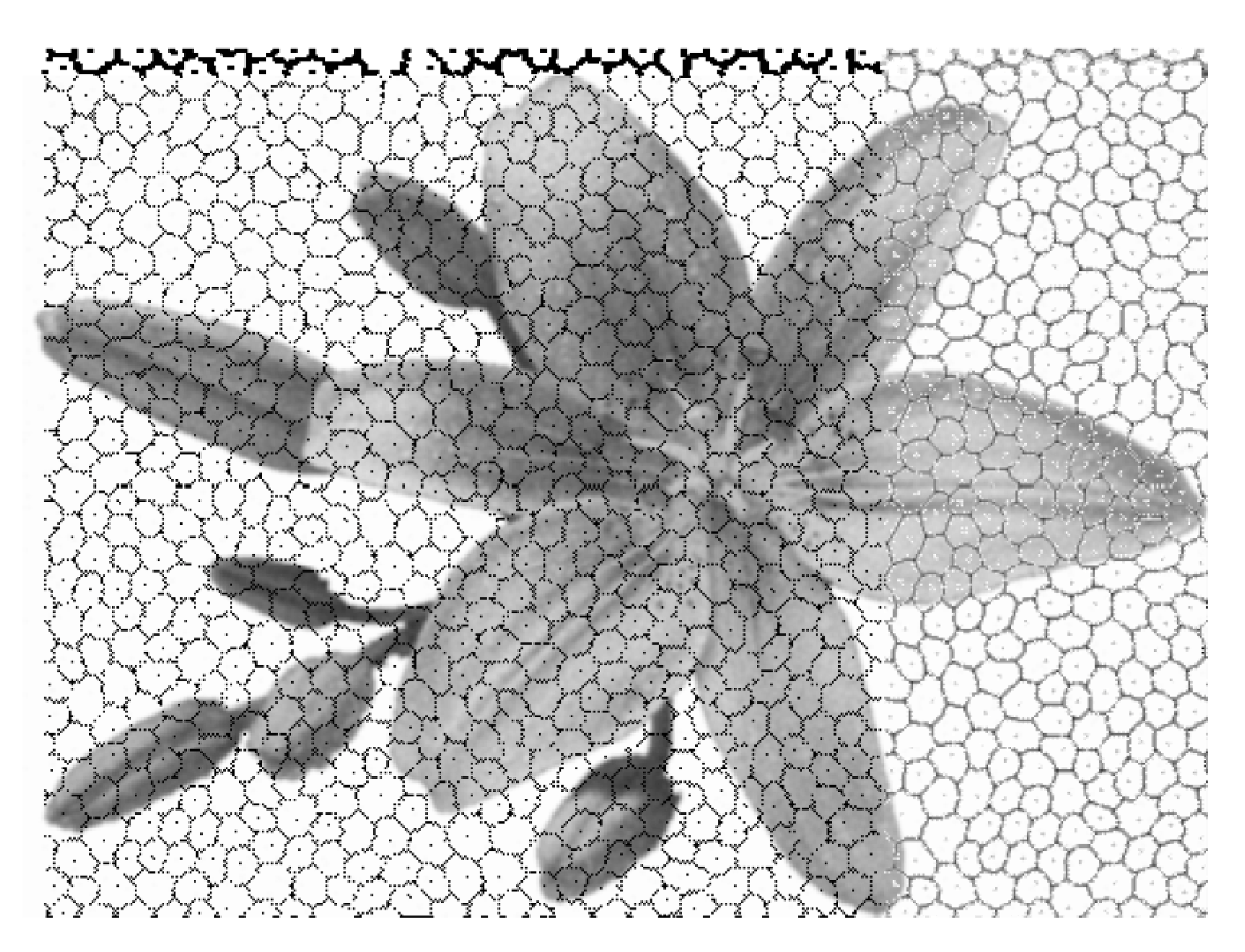


图 2.29　产生的随机聚合网屏叠加在图像上

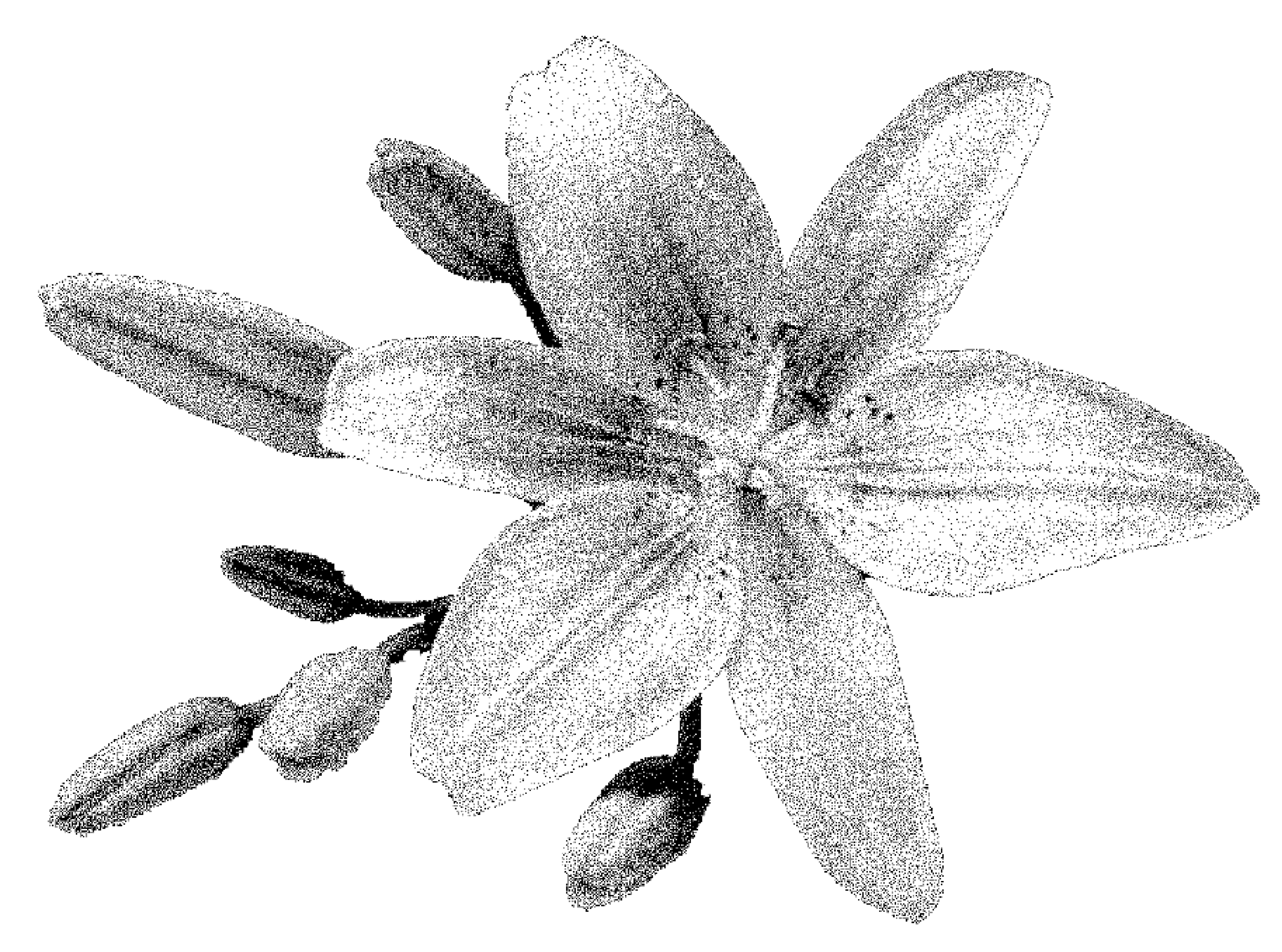


图 2.30　网屏输出的结果图

2.3.2　基于 GPU 的半色调图像生成

上面的算法在对网点中心进行 Delaunay 三角剖分后，可通过 Voronoi 剖分得到每个中心网点的最大覆盖区域，但其并没有采用直接的 Voronoi 剖分，这些方法得到的网点最大覆盖区域是不精确的。本节介绍一个基于 GPU 的半色调图像生成算法[Qi2007]，结果更精确，且可越过三角剖分过程，直接得到 Voronoi 剖分，速度上大大提高。

该算法首先构造了一个大的抖动矩阵，并随机放置网点，然后利用 GPU 将其 Voronoi 剖分，使抖动矩阵中的每个中心网点都获得它的最大覆盖区域。最后利用 GPU 对每个网点的覆盖区域进行灰度填充，得到一个大的阈值矩阵，也就是随机聚合网屏。

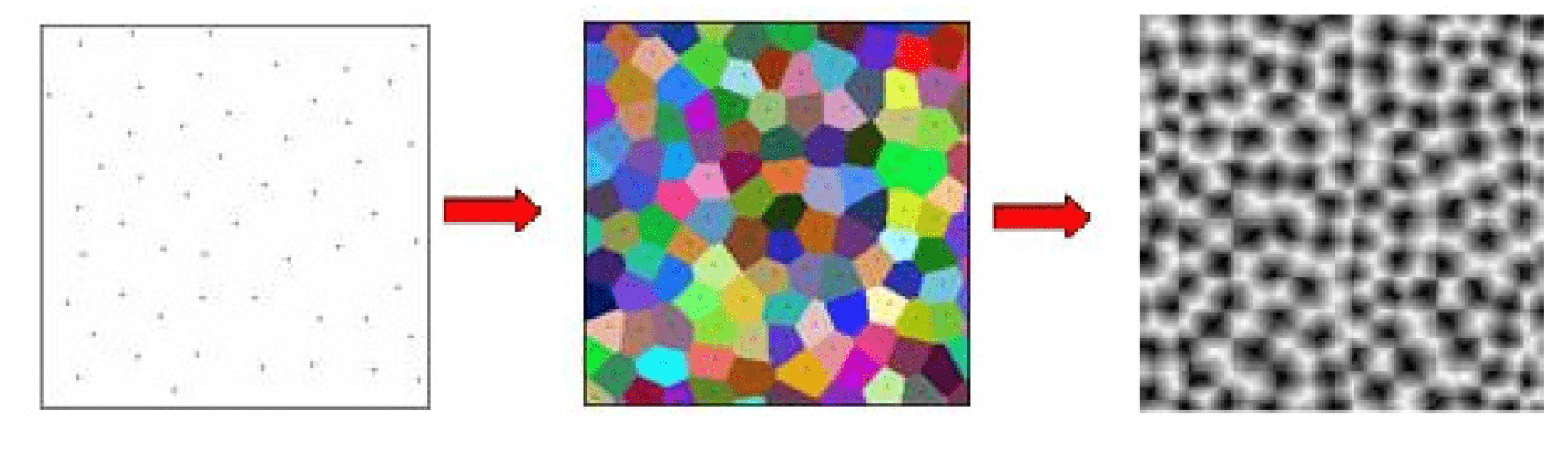
类似 2.3.1 节中的算法，本算法过程如图 2.31 所示，也包括三个主要步骤。

（1）构造抖动矩阵，随机布置中心网点。

（2）基于 GPU 计算中心网点的 Voronoi 图。

（3）基于 GPU 对每个中心网点的最大覆盖区域进行灰度填充。

步骤（1）采用 2.3.1 节介绍的方法构造一个大的抖动矩阵，并随机放置网点，这里不再重复。下面重点介绍步骤（2）和步骤（3）。



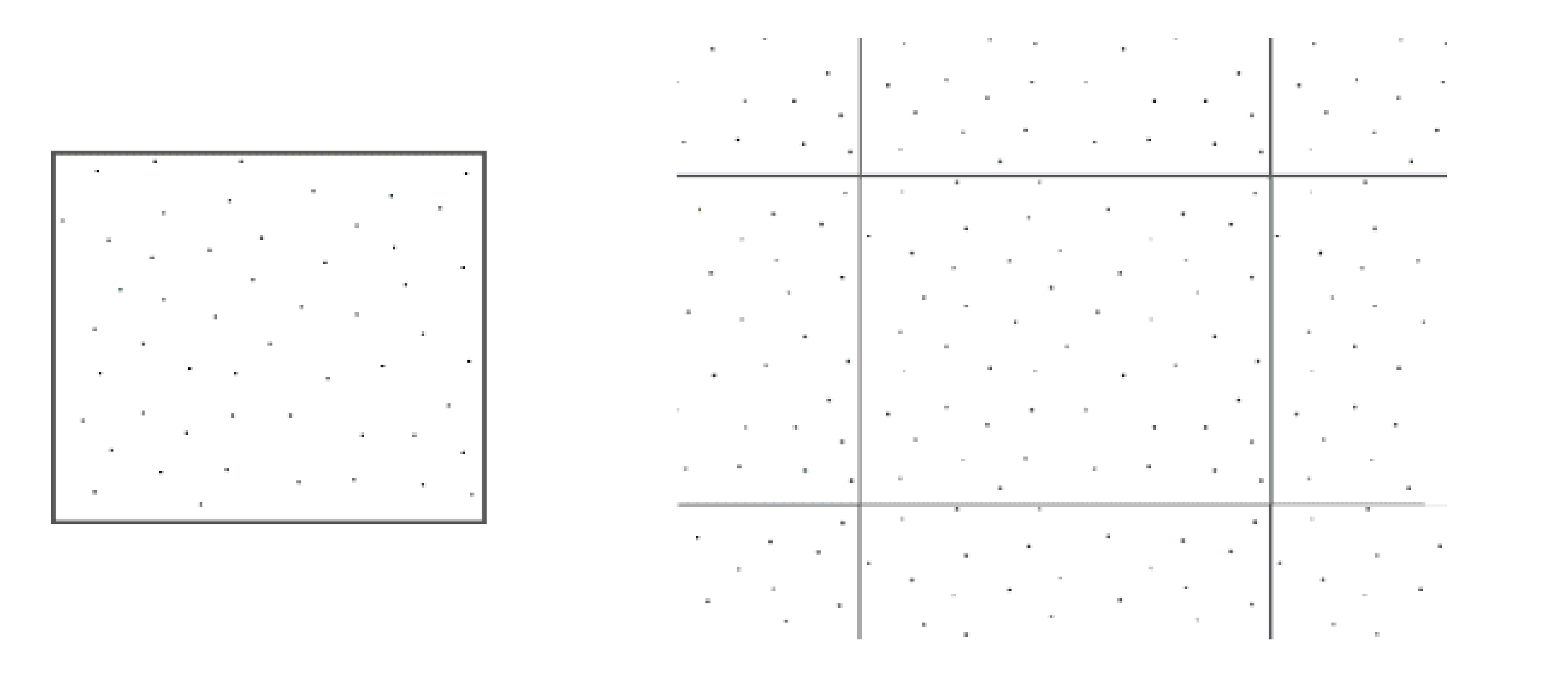
(a) 随机布置中心网点　　(b) 计算中心网点的 Voronoi 图　　(c) 灰度填充

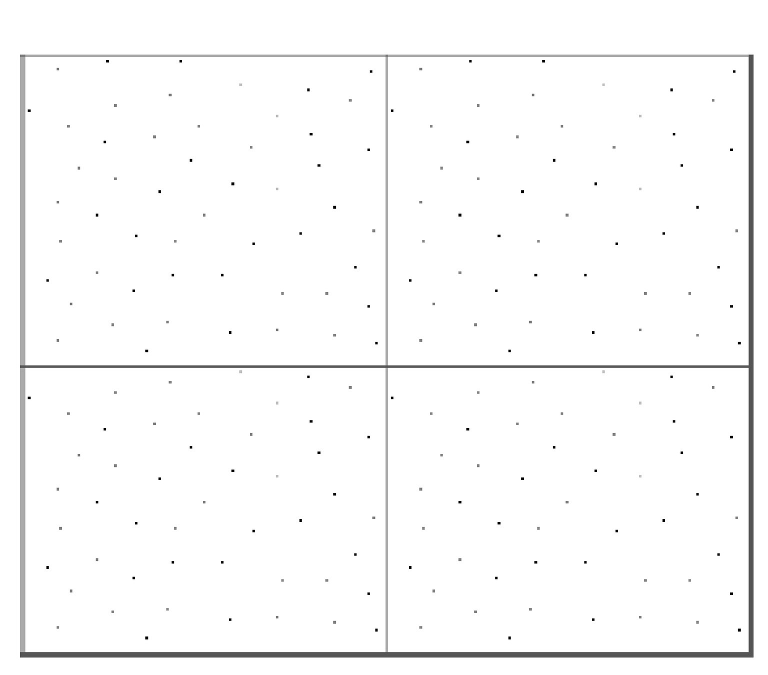
图 2.31　算法步骤示意图

1. 基于 GPU 的网点 Voronoi 剖分

在步骤（1）中，我们在大的抖动矩阵中放置了中心网点，接下来需要得到每个中心网点的最大覆盖区域，以便于下一步的灰度填充。为了保证抖动矩阵在边界处的连续，在进行 Voronoi 前需要对矩阵进行补点。把抖动矩阵上面 1/3 的部分补到矩阵下边，把抖动矩阵下面 1/3 的部分补到矩阵上边，把抖动矩阵左面 1/3 的部分补到矩阵右边，把抖动矩阵右面 1/3 的部分补到矩阵左边。在矩阵的左上角、右上角、左下角、右下角也进行同样的补充，补点效果如图 2.32 所示。

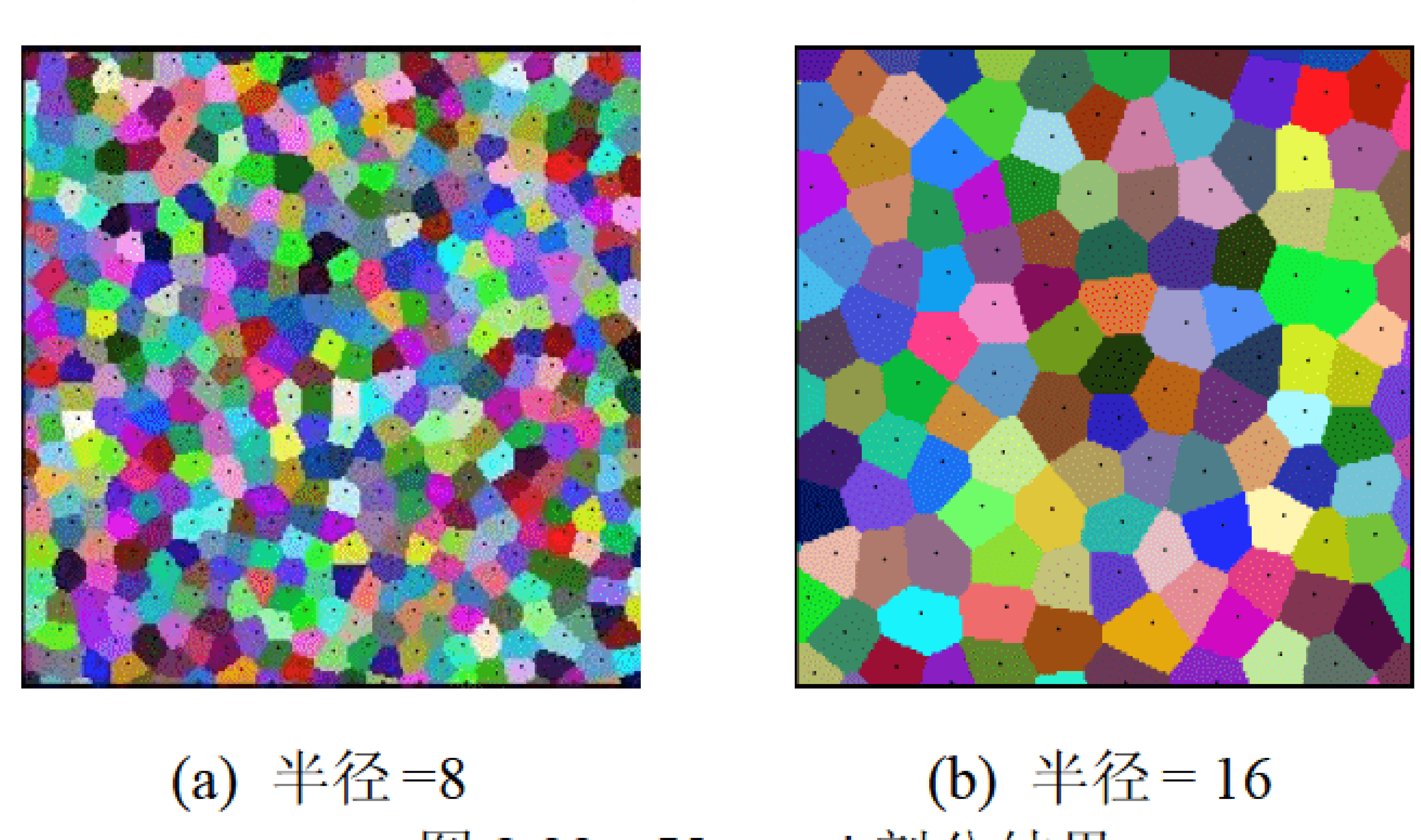
通过 Voronoi 剖分可以得到每个中心网点的最大覆盖区域。算法采用了 2.2.4 节介绍的基于 GPU 的 Voronoi 剖分的方法。图 2.33 分别显示了半径 r =8 和 r =16 时中心网点 Voronoi 剖分的结果。





(a) 抖动矩阵　　(b) 补点后的抖动矩阵　　(c) 四方连续

图 2.32　构造抖动矩阵，随机布置中心网点



(a) 半径 =8　　(b) 半径 = 16

图 2.33　Voronoi 剖分结果

经过这一步，我们已经得到每个中心网点的最大覆盖区域，下一步将对每个最大覆盖区域进行快速、高质量的灰度填充。

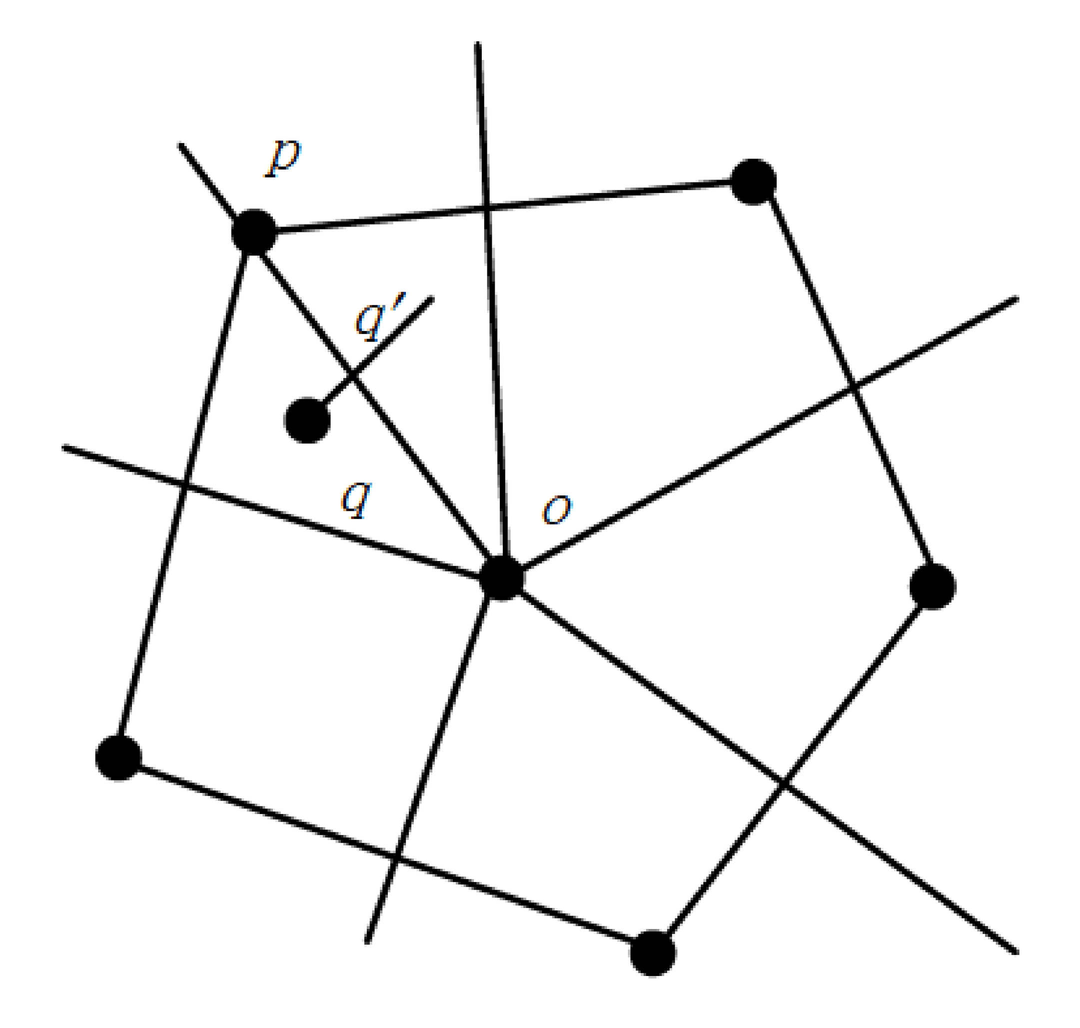
2. 基于 GPU 的灰度填充

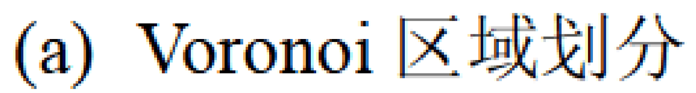
通过前两步的处理，我们已经得到了网点的分布和区域。最后我们要做的就是对每个中心网点的覆盖区域进行灰度填充。这里，我们采用了一种基于 GPU 的比例填充模式，可以在常数步内完成所有的填充任务，由于 GPU 的特性，使得实际运行速度有了成倍的提高，且不需要后期的平滑处理，在绘制质量上也有了明显的提高。

我们以一个 40×40 的抖动矩阵为例，如图 2.34（b）所示，其中的白色圆点代表了网点中心，不同的灰度块代表了不同网点中心的 Voronoi 区域。对于任一个 Voronoi 区域（见图 2.34（a）），我们将其分成几个部分，对于每一个小部分，由网点中心向边界角点按照灰度级别由小到大填充起来。整个填充过程包括如下几步。

（1）找角点边界

通过对图的观察，我们可以得到角点的判定信息。对于一个点，如果与之相邻的 8 个点中至少存在两个不与之同站点的邻居，那么它就是角点。从图 2.34 中直观地看，如果有两个或两个以上与自己不同灰度的邻居点，那么这个点本身就是角点。点$(x，y)$的 8 个邻居点定义为$(x+k，y+k)$，$k=\{-1，0，1\}$。此过程只需要运行一个顶点程序和一个片段程序，经过一次计算就可以完成，复杂度是 $O(1)$。




(a) Voronoi 区域划分

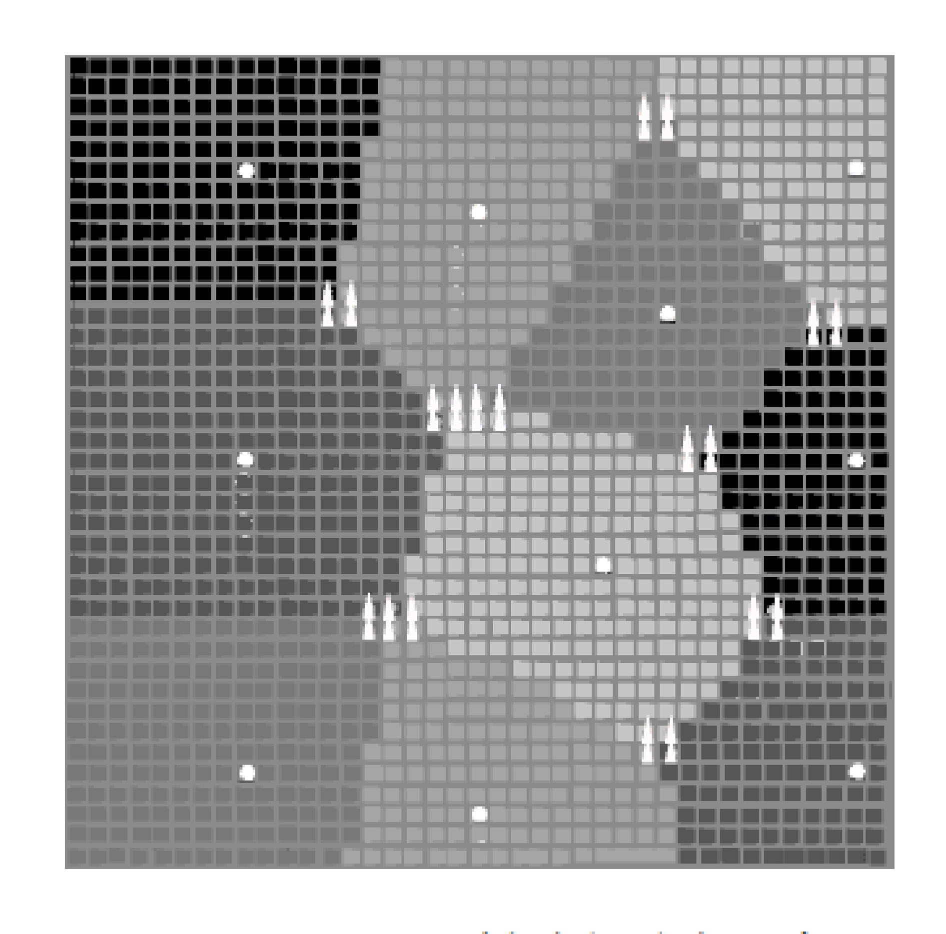

(b) 40×40 的抖动矩阵

图 2.34　网点中心的 Voronoi 图与 Voronoi 区域的划分

（2）确定每个点对应的角点

步骤（1）获取的角点可能会有多重选择，如图 2.34（b）所示，图中的白色标注为三角形的点都是角点，那么对应于一个边界角，即三种颜色相邻处可能有不止一个角点。对于任意一点 q，要想计算它的灰度值，我们需要找到它所对应的站点 o 和角点 p。对于每个点，它所属的站点通过步骤（2）的 Voronoi 剖分已经找到，因此仅需要去找它对应的角点。显然，每个点所对应的角点一定跟自己属于同一个站点，也就是颜色一样，而两个相邻站点间的距离是 r 到 $2r$，因此在进行 Flooding 找对应角点时，最后一轮的 step 大小为 r，整个过程可在 r 步内结束，所以时间复杂度是 $O(1)$。当然在通过 Flooding 找角点的过程中，由于角点可能不唯一，所以需要确定一个唯一的角点，但这也只须在片段程序中添加一行比较表达式就可以做到，不需要增加额外的 Flooding 轮数。

（3）灰度计算

通过步骤（2），对于任意一点 q，已经找到它所对应的站点 o 和角点 p，作 q 到 op 的垂线，得到垂足 q'，则利用 oq' 与 op 的距离比，我们就可以得到点 q 处的灰度值。

$$q_{gray} = (distance\,(oq') \,/\, distance(op)) * 255$$

至此整个灰度填充过程就结束了。图 2.35 由左到右分别是采用 2.3.1 节与本节方法生成的二值图像。相比之下，本节方法生成的图像细节层次更丰富，颜色过渡更自然，在高光处理上（鼻子、帽檐）也更理想一些，图像更加细腻（胡须、头发）。

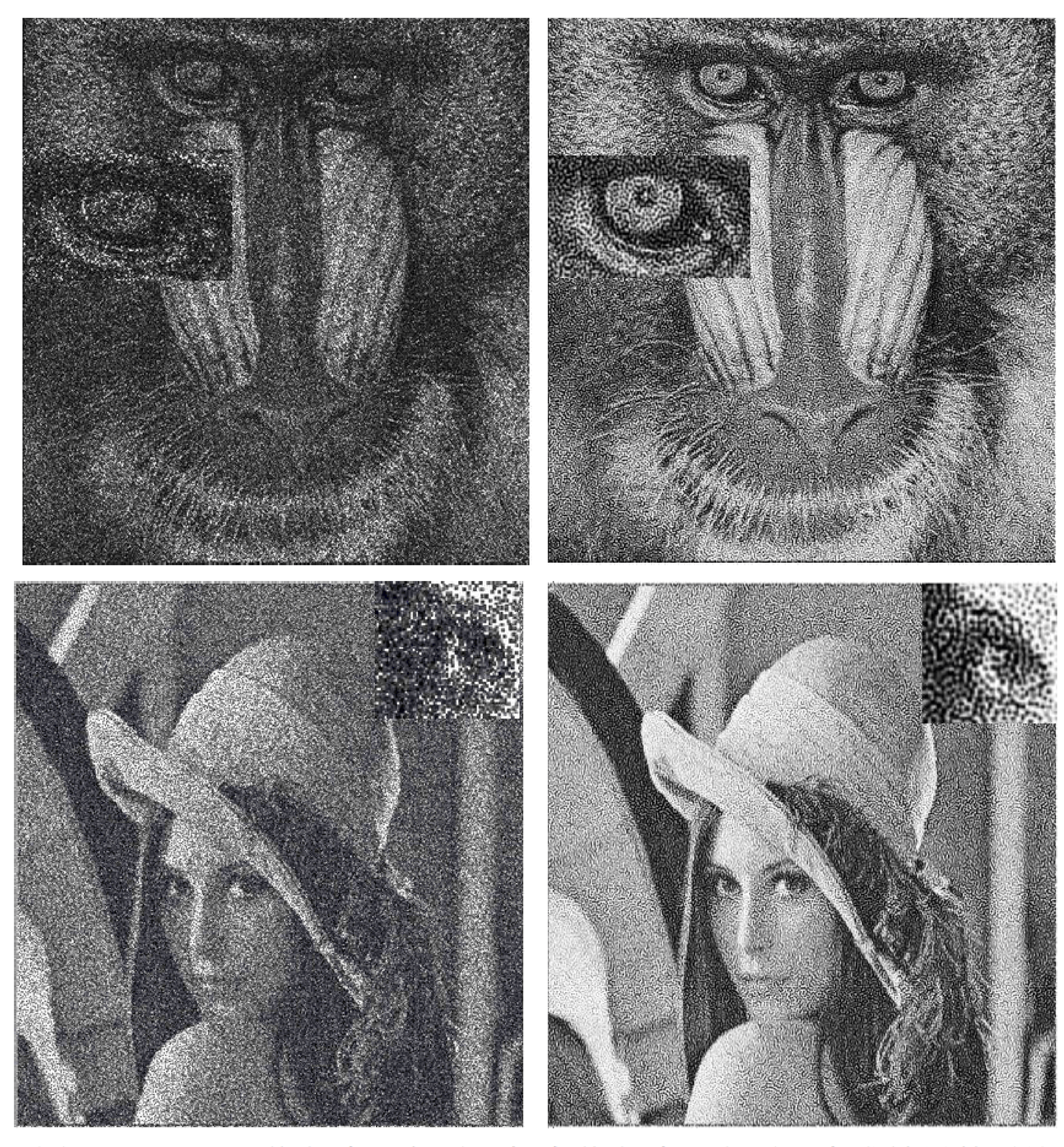


图 2.35　2.3.1 节方法（左列）与本节方法（右列）生成效果的对比

2.3.3　带状图像的骨架计算

在自动文字识别、指纹识别、染色体分析和自动线路板检测等识别系统中，采用细化方法计算图像的骨架是一个十分重要的预处理步骤。因为骨架包含了文字图像特征的最有效数字化信息，能对文字图像进行有效的描述，迄今为止已有很多细化算法产生，大致可归纳为迭代算法和非迭代算法两大类。迭代算法主要是根据骨架的特性来制定一些限制条件，通过由外到里逐步去除边缘点来求得图像的骨架。但在一般情况下迭代算法的速度比较慢。非迭代的细化算法要比迭代算法快得多，因为它们是在一个和分辨率无关的固定数目的步骤下完成。这类算法获取骨架的最常用技术是利用图像距离的

变化来求得可能组成骨架的像素，然后根据骨架的特性来选择其中的子集。其中许多算法都是基于区域边界的逼近多边形进行骨架计算。关于骨架计算的相关工作请参见文章[YangCL2000，YangYJ2002]。本节主要介绍一种基于 Delaunay 三角剖分的快速细化算法[YangYJ2002]，可用于文字、工程图、指纹等类型的带状图像的骨架计算。为便于算法描述，假设图像是二值图像，且已经进行了一些噪声处理。

算法首先计算带状图像边界的多边形；然后基于边界多边形的顶点进行 Delaunay 三角剖分，并对生成的三角形进行调整，从而得到该边界多边形的比较理想的三角剖分结果；接着，根据每一个三角形的拓扑结构来计算局部骨架，最后连接这些局部骨架，就构成了整个文字图像的骨架。对于多边形存在内边界的情况，算法需要先把有内边界的文字图像的某一部分切断，形成一个没有内边界的图像，然后再按无内边界的情况进行处理。

1. 图像边界多边形的计算

算法首先计算图像的边界多边形。由于边界多边形的边的长度变化非常大，为了产生一个更精确的骨架，有必要用文字图像的宽度（用 d 表示）对边界多边形的边进行分割，形成较短的边。那么，对于每一个文字都需要估计一个参数 d。由于大部分文字都包括大量的垂直或水平笔划，所以可以通过水平和垂直扫描来估计文字笔划的宽度。

在图像扫描过程中，扫描线与图像相交得到一系列线段，每一条线段就叫一个游程。可用一个频率数组来记录所有游程出现的次数。其中，频率数组的下标对应游程的长度，频率数组的元素就记录不同长度的游程出现的次数。在图像扫描过程中，每得到一个游程，就使相应频率数组元素的值加 1。在整个扫描过程结束后，频率数组中值最大的元素的下标就为该图像的宽度 d。

一旦带状图像的宽度被估计出来，边界多边形上所有长度大于 d 的边被分成许多长度为 d 的小边。如果分割所得的最后一条边的长度小于 d，则把它合并到上一条边中。

2. 不含内边界的多边形的 Delaunay 三角剖分

算法首先以边界多边形的所有顶点为站点，采用 2.2 节介绍的算法进行离散点集的 Delaunay 三角剖分；由于生成的点集的三角形有可能和多边形相交，所以在三角剖分以后对生成的三角形进行处理，以保证最后生成的三角形完全在多边形内部或者外部；然后根据三角形的面积把位于多边形外部的三角形去掉；最后对剩余（在多边形内部）的每个三角形根据拓扑结构细化成一条边或者三条边，再连接而成带状图像的骨架。为了方便处理，我们按逆时针方向对边界多边形上的顶点由小到大进行编号。算法生成以下三种类型的 Delaunay 三角形。

- 内部三角形：Delaunay 三角形完全在多边形内部；
- 外部三角形：Delaunay 三角形完全在多边形外部；
- 相交三角形：Delaunay 三角形和多边形相交。

由于最终分割边界多边形的所有三角形必须在该多边形的内部，所以对于以上三种类型的三角形来说，所得的内部三角形是最终结果的一部分，应当保留；外部三角形完全在边界多边形外部，应当删除；对于相交三角形，由于和边界多边形相交，需要把其剖分成一些完全在多边形内部的三角形和一些完全在多边形外部的三角形，使得剖分以后的三角形或者在多边形内部，或者在多边形外部。

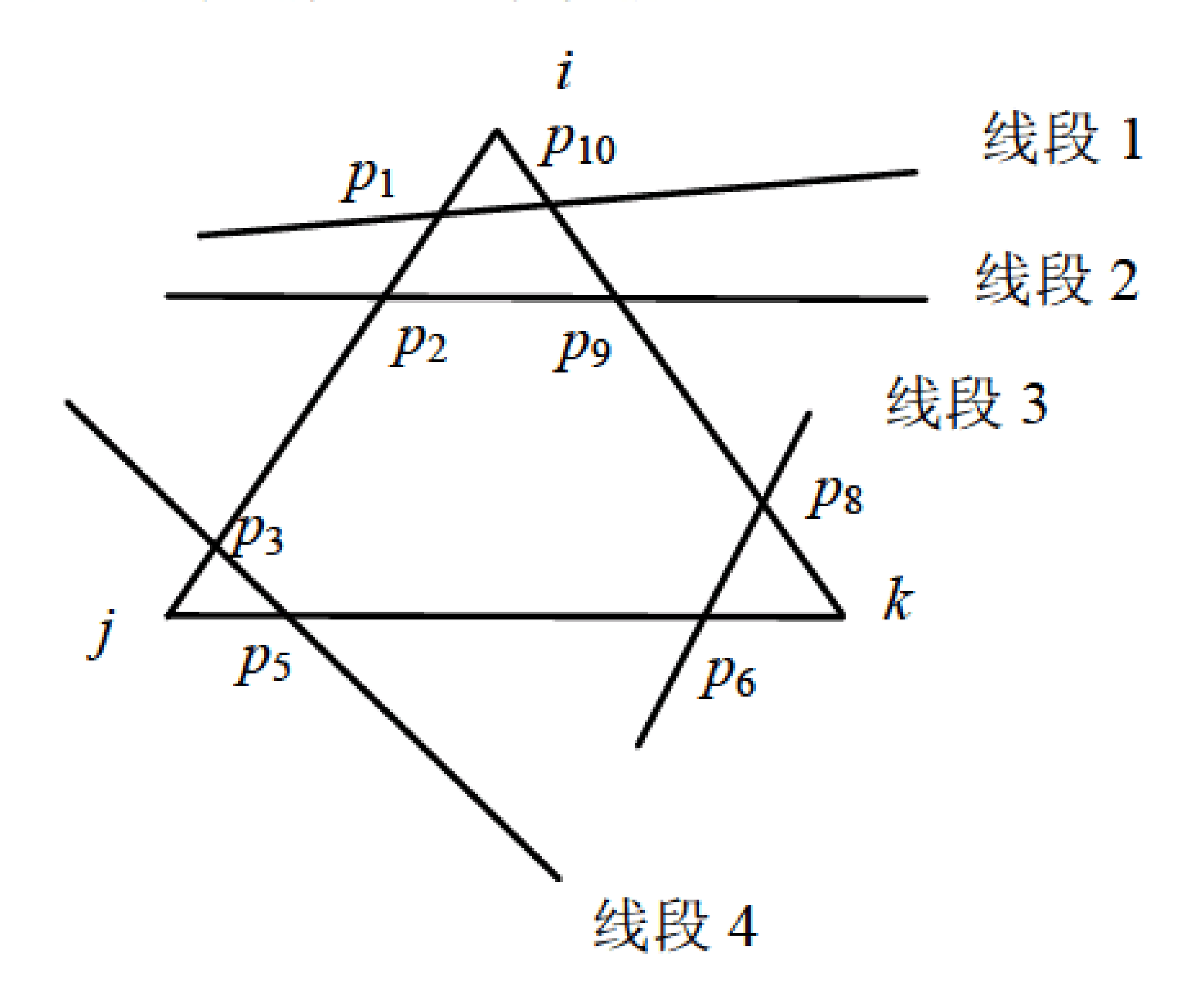
需要特别说明的是，由于前面已使用带状图像宽度把边界多边形的边分割得比较短，所以 Delaunay 三角剖分后得到的大多数都是内部三角形或外部三角形，相交三角形很少，所以需要进一步剖分的三角形很少，对整个算法速度影响较小。

3. 相交三角形的剖分

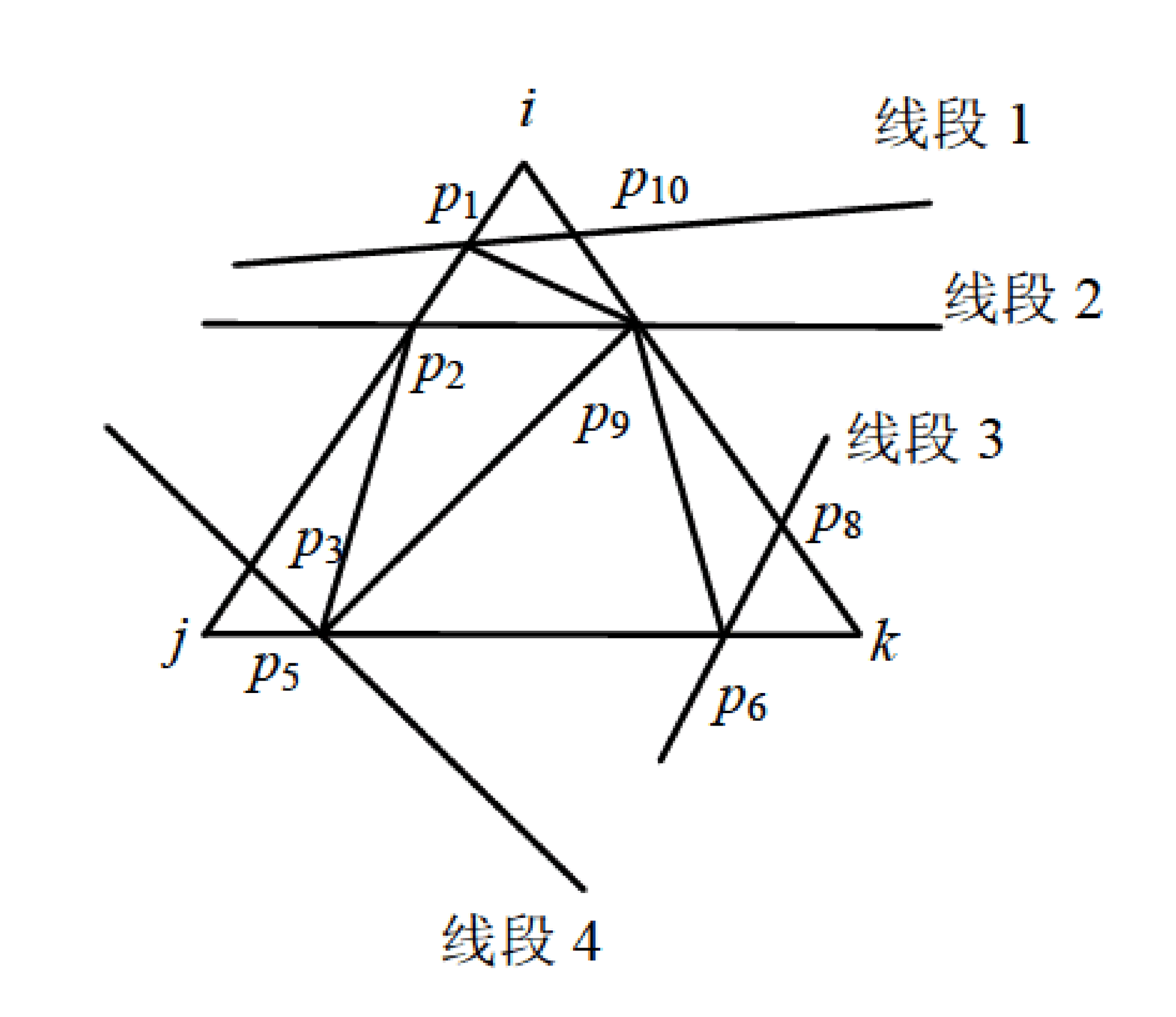
由于多边形的边不会自交且 Delaunay 三角形互不覆盖，所以 Delaunay

三角形与边界多边形相交的情况一般如图 2.36（a）所示。

假设一个 Delaunay 三角形的顶点分别为 i、j、k。首先计算三角形和边界多边形的交点，并按照 $i \rightarrow j \rightarrow k \rightarrow i$ 的顺序放到一个数组里。这样数组中点的存放顺序为 i→在 i 和 j 之间的交点→j→在 j 和 k 之间的交点→k→在 k 和 i 之间的交点→i 点。所以，在该数组中存放的点有两类：Delaunay 三角形的端点、Delaunay 三角形与边界多边形边的交点。后者可以直接通过边界多边形的边到达另一个交点。



(a) 与边界多边形相交的 Delaunay 三角形



(b) 对图(a)中的 Delaunay 三角形进行剖分后生成的三角形

图 2.36 与边界多边形相交的 Delaunay 三角形及其剖分

下面以图 2.36（a）为例，介绍一般相交三角形的剖分方法。首先，以 $i \rightarrow p_1$ 作为起始边，从 p_1 点通过线段 1 到达 p_{10} 点，从 p_{10} 点继续到达 i 点。这样得到一个凸的多边形 ip_1p_{10}，然后可以很容易把它进行 Delaunay 三角剖分，并将生成的三角形添加到三角形序列里。这时，该凸多边形中只有边 $p_1 \rightarrow p_{10}$ 是新边（不位于三角形边界上，其余的边都位于三角形边界上），接着以 $p_1 \rightarrow p_{10}$ 为起始边，重复上面操作，完成其余部分的剖分。

为了便于删除外部三角形，赋予每一对多边形的边和三角形的边所形成的交点对应的边界边端点序号。对图 2.36（a）进行剖分和 Delaunay 三角剖分处理以后，生成的三角形如图 2.36（b）所示。

4. 外部三角形的删除

对相交三角形进行处理后，生成的所有三角形或者在边界多边形的内部，或者在边界多边形的外部。为了删除外部三角形，需要利用三角形面积的正负概念进行判断：如果三角形的顶点顺序是逆时针方向，则三角形面积为正；如果是顺时针方向，则三角形面积为负。

由于边界多边形的顶点是按逆时针由小到大的顺序进行编号的，对其三角剖分后得到的三角形顶点也都有编号。不难发现，如果对于一个三角形按其顶点编号由小到大的顺序走动，则符合以下规律。

- 内部三角形的走动方向为逆时针方向，计算出的面积为正；
- 外部三角形的走动方向为顺时针方向，计算出的面积为负。

因此，对于所有的三角形，如果其面积为正，则表明此三角形为内部三角形，应保留；如果三角形的面积为负，则为外部三角形，应删除。图 2.37（a）和图 2.37（b）给出了字母“C”的三角剖分结果。

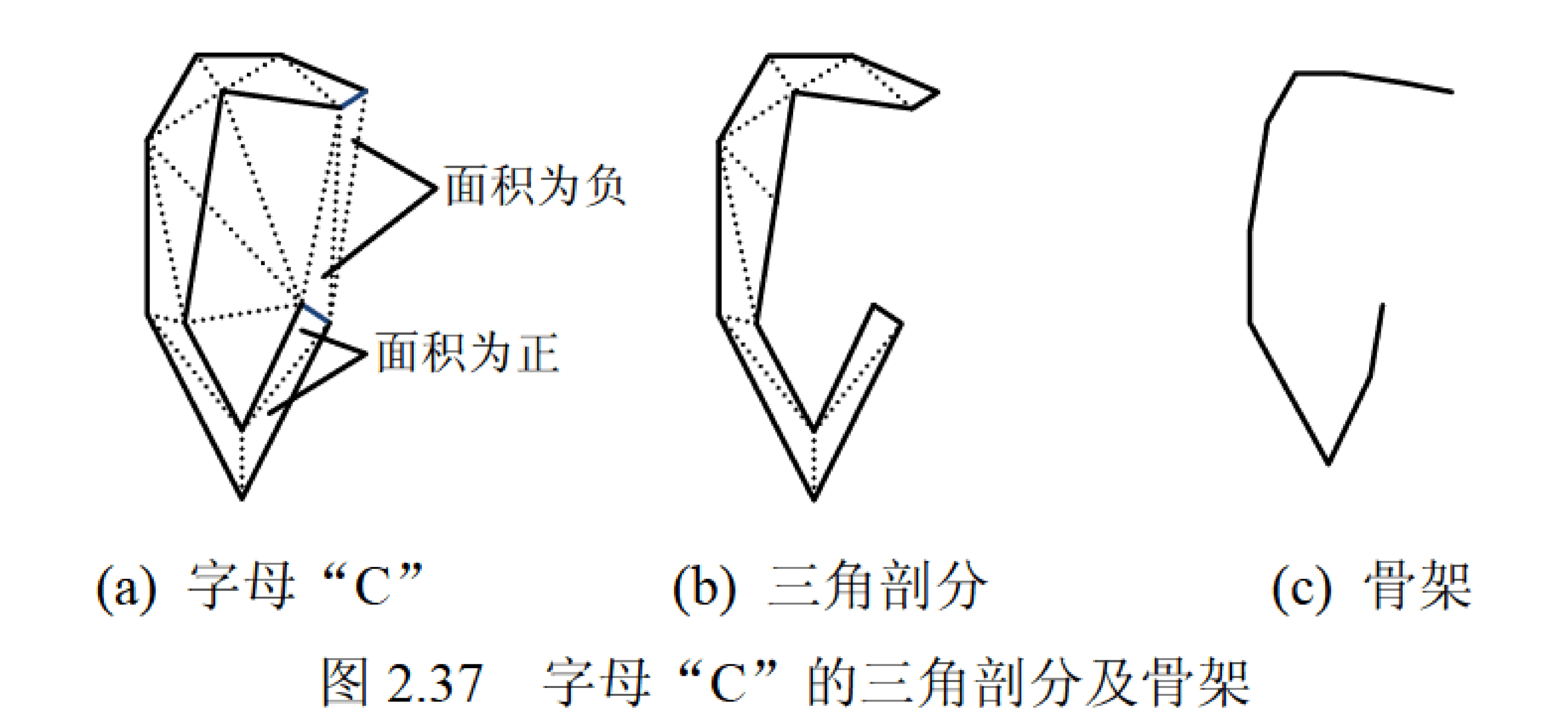


图 2.37 字母“C”的三角剖分及骨架

5. 骨架计算

对于最终得到的三角形，把其在多边形边界上的边称为外部边，在多边形内部的边称为内部边。三角形三条边的类型不同，决定了三角形在整个图像中的位置特点。归纳起来，一般分为以下三种类型。

- 终端三角形：具有两个外部边；

- 连接三角形：具有一条外部边；
- 分支三角形：没有外部边。

这三类三角形的骨架定义如下。

- 终端三角形：三角形内部边的中点和对应顶点的连线（见图 2.38（a））；
- 连接三角形：三角形两条内部边中点的连线（见图 2.38（b））；
- 分支三角形：重心和三条边中点的连线（见图 2.38（c））。

图 2.37（c）给出了字母“C”的细化结果。

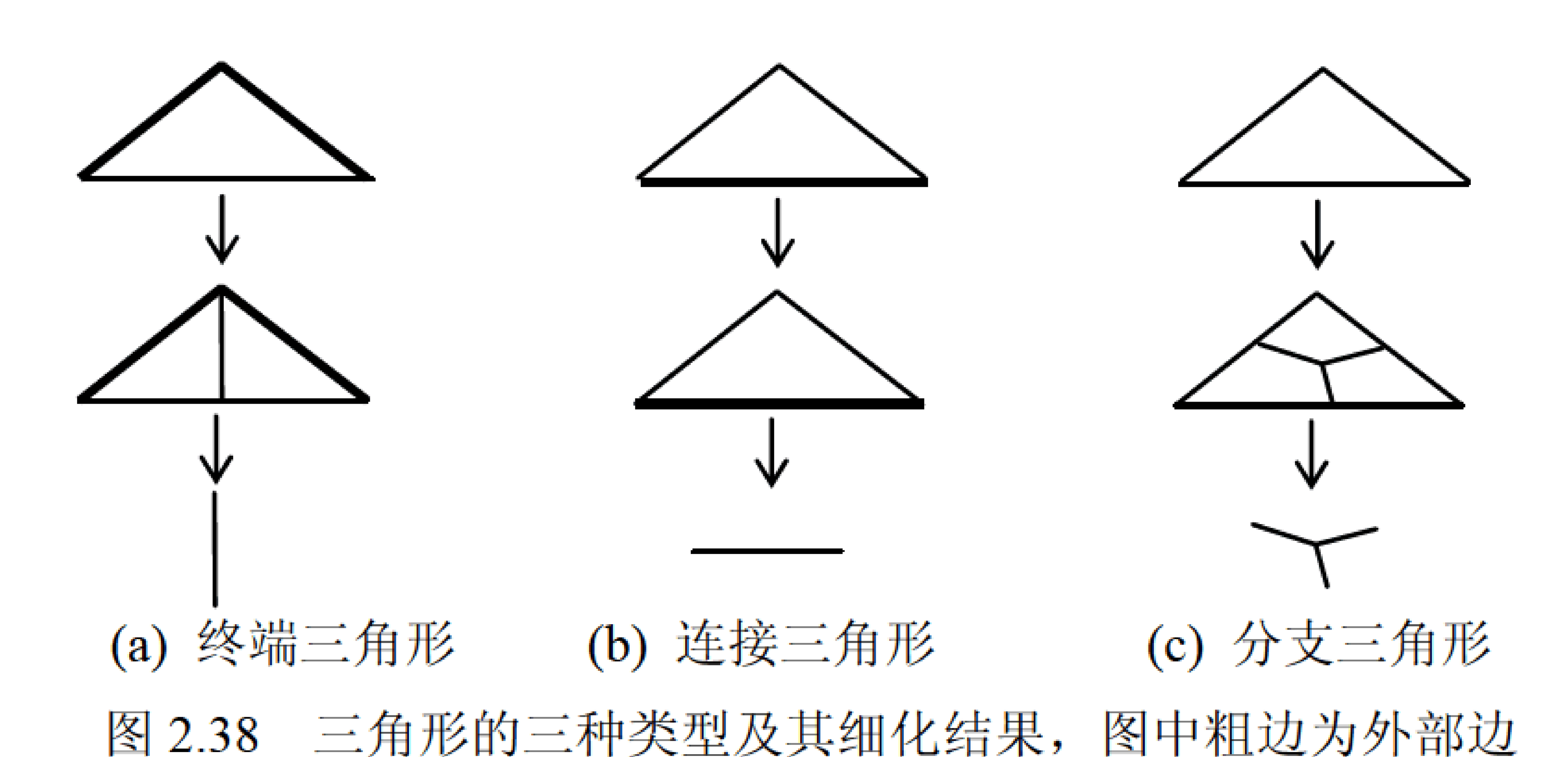


图 2.38　三角形的三种类型及其细化结果，图中粗边为外部边

6. 含内边界的多边形的三角剖分

实际上，很多带状图像边界的多边形有时包含多个内边界。可以先去掉所有内边界，再按前面所介绍的算法进行骨架计算，最后再去除毛刺和恢复交叉点。

在第 4 章中，我们还将介绍计算骨架的其他算法，可直接对含内边界的多边形进行三角剖分和骨架计算。

第 3 章　多边形的 Voronoi 图及其应用

本章介绍多边形的 Voronoi 图，包括其概念、性质、构造方法及若干应用。

3.1　定义与性质

3.1.1　定义

在离散点集的 Voronoi 图中，站点只有离散点，Voronoi 边只来自于两个离散点之间的中分线。但在多边形中，站点包括多边形的顶点和边，它们之间的中分线主要存在以下四种情况。

（1）点与点：其中分线是它们之间连线的垂直平分线，该线将平面一分为二，分别是距离这两个点的最近点的集合。

（2）点与边（点是该边的一个顶点）：其中分线是过所给点且垂直于给定边的直线，该线也将平面一分为二，分别是距离给定点和边的最近点的集合。

这里，点到边的距离是指：如果该点的垂足在边上，则该点到边的距离为该点到其垂足的距离，否则，该点到边的距离为该点到边的顶点中最近的那个距离，如图 3.1 所示。

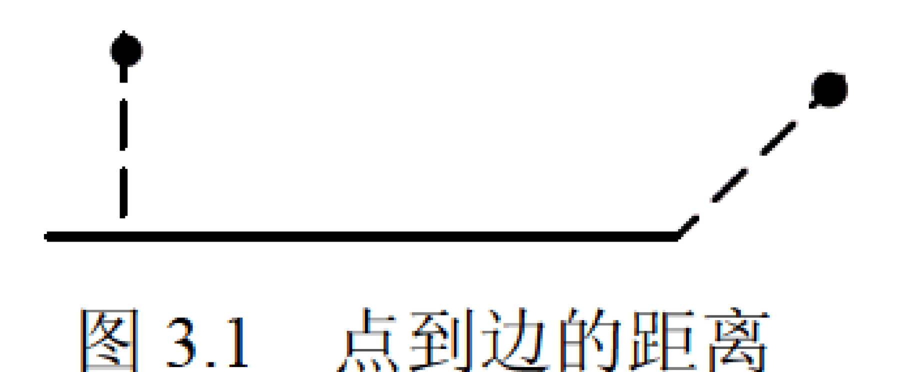


图 3.1　点到边的距离

（3）点与边（点不在该边上）：中分线是一条双曲线，该线上的点到给定点和边所在直线的距离相等。

（4）边与边：中分线是两边所形成夹角的角平分线。特殊情况下，若两边平行，则其中分线为平行于这两条边且距离相等的中线。

由此可见，多边形的两个站点 p_1 和 p_2 的平分线为到 p_1 和 p_2 距离相等的点轨迹，用 $b(p_1, p_2)$表示。半平面 $h(p_1, p_2)$是指平面上到站点 p_1 比 p_2 的距离更近的所有点的集合。

令 P 是多边形的所有站点集合，对于任意站点 $p_i \in P$，其 Voronoi 区域是指平面上到 p_i 的距离比 P 中任何其他站点距离更近的所有点的集合。它等价于平面 $h(p_i, p_j)$的交，其中 $p_j \in P-\{p_i\}$。因此，其可表示为：

$$\mathrm{VR}(p_i)=\cap\{h(p_i, p_j) | p_j \in P-\{p_i\}\}$$

令 P 是多边形的所有站点的集合，多边形的 Voronoi 图表示为：

$$\mathrm{VD}(P)=\cup\{\mathrm{VR}(p) | p \in P\}$$

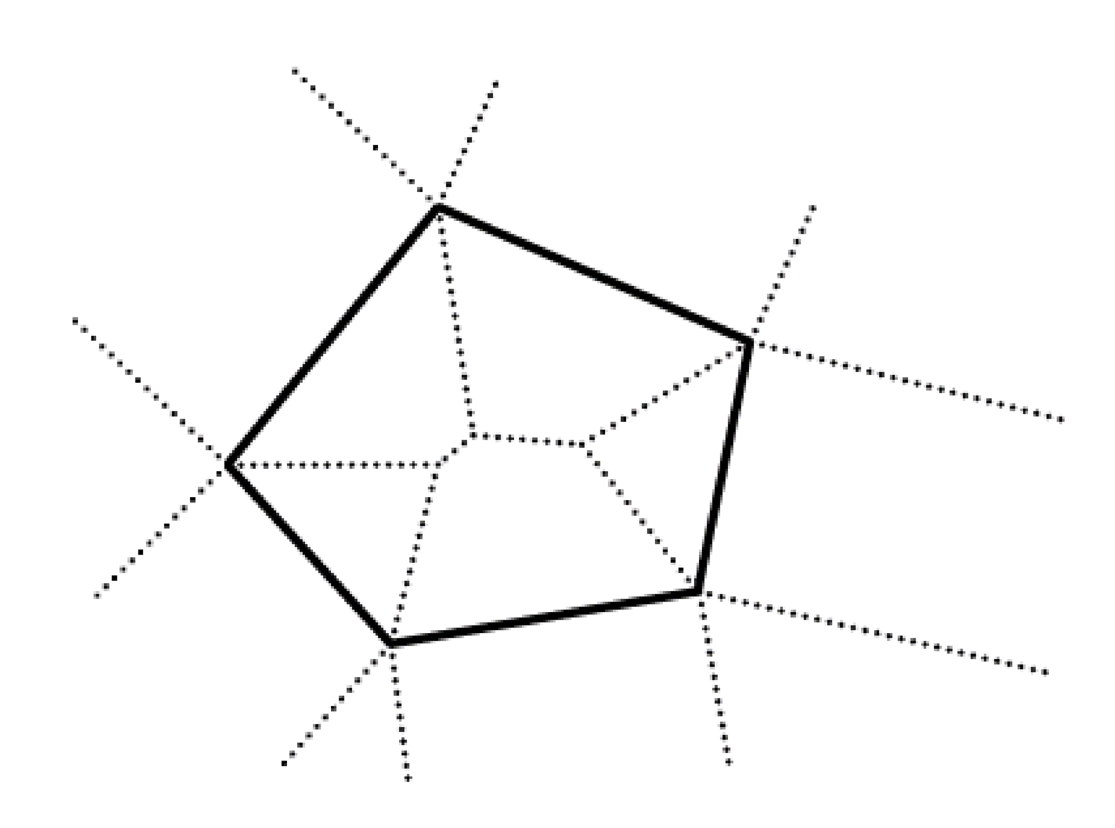


图 3.2　多边形的站点形成的 Voronoi 图，其中虚线是 Voronoi 边

上面的定义实际上是多边形的顶点和边形成的平面直线图的 Voronoi 图（如图 3.2 中虚线部分）。在实际应用中，常用到的多边形的 Voronoi 图实际上是对多边形域中的 Voronoi 剖分。在第 1 章中我们定义了两类多边界多边形。

（1）对于一类多边界多边形，有一个最外面的边界，称之为外边界；其他边界都在它的内部，称之为内边界（如图 3.3（a）中实线部分）。

（2）对于另一类多边界多边形，它不存在包含内边界的外边界（如图 3.3（b）中实线部）。

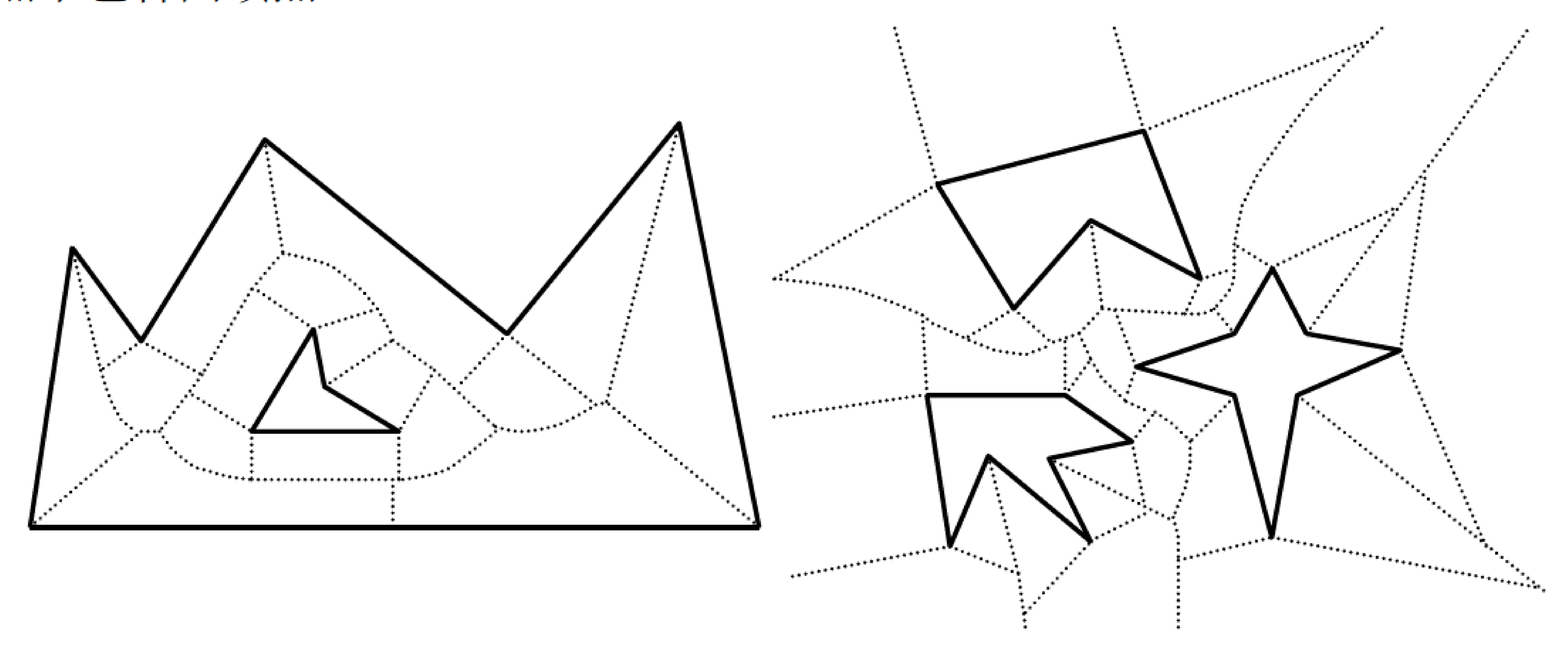
对于这两类多边形，假设外边界是逆时针（顺时针）的，内边界是顺时针（逆时针）的，如果一个人沿这些边界走动，多边形总是位于每条边界的左侧（右侧）。

相应地，对于多边界多边形的 Voronoi 图也分为两类。

（1）对于第一类多边界多边形，其 Voronoi 图位于外边界的内部和所有内边界的外部，并将这部分平面区域分割成许多单元。我们将这类多边界多边形的 Voronoi 图称为内部 Voronoi 图。

（2）对于第二类多边界多边形，其 Voronoi 图位于所有边界的外部，并将这部分平面区域分割成许多单元。我们将这类多边界多边形的 Voronoi 图称为外部 Voronoi 图。

图 3.3 给出了这两类多边界多边形（实线部分）和它们的 Voronoi 图（虚线部分）。在这两类多边界多边形的 Voronoi 图中，只有边和内尖点的 Voronoi 区域非空。因此，我们提及这两类多边界多边形的 Voronoi 图时，所说的站点不包含凸顶点。



(a) 第一类多边形与内部 Voronoi 图　　(b) 第二类多边形与外部 Voronoi 图

图 3.3　多边形及其 Voronoi 图

3.1.2 性质

本节介绍多边形的 Voronoi 图的性质[YangCL2005，YangCL2006c]。为便于描述，下面给出一些符号表示：

- h 表示多边形 P 所含的内边界的数目；
- n、k、s 分别是 P 的所有顶点、内尖点和凸顶点的数目，其中 n=k+s；
- t、r 分别是位于 P 的凸包上的所有顶点和边数；
- m、e 分别是 VD(P)的所有 Voronoi 顶点和边数；
- a_v、a_e 分别表示每个 Voronoi 区域包含的 Voronoi 顶点和 Voronoi 边的平均数。

1. 多边形的内部 Voronoi 图的性质

引理 3.1 令 j、i 分别是一棵树中的非叶结点和叶结点的数目。如果每个非叶结点至少有 2 个儿子，且根结点至少有 3 个儿子，则：

$$j \leqslant i - 2$$

特别地，如果每个非叶结点有且只有 2 个儿子，且根结点有且只有 3 个儿子，则有：

$$j = i - 2$$

引理 3.2 对于多边形 P 的一条内边界 B，VD(P)中必存在一个围绕 B 的环。

证明：VD(P)中必存在一条包围 B 的封闭的线，P 中在该线内侧的点距离 B 比距离 P 中其他任何站点更近。这条封闭的线便是 VD(P)中围绕 B 的环。证毕。

引理 3.3 对于一个有 h($h \geqslant 0$)个内边界的多边形，其内部 Voronoi 图有

h 个环。

证明： 一方面，假设内部 Voronoi 图中的环多于 h 个。设其中一个环的内部没有内边界，且这个环没有边或顶点与多边形的站点相关联。由于这个环所包围的区域与这个多边形的任何边界站点都不关联，所以这与多边形的 Voronoi 图的定义矛盾。另一方面，根据引理 3.2 知内部 Voronoi 图中的环不会少于 h 个。命题成立。证毕。

定理 3.1　对于多边形的内部 Voronoi 图，有：

$$m \leqslant n+k-2+2h$$

证明： 如果多边形有内尖点或内边界，则其内部 Voronoi 图存在一些环。为了由其内部 Voronoi 图构造一棵树，我们首先将每个内尖点分成两个顶点，并添加一条短线段连接它们，分别作为修改后的多边形的顶点和边。这样就可以消除内部 Voronoi 图在内尖点处的环。然后，在每个环的一条 Voronoi 边的中点处将其剪断，且在剪断处得到两个剪断点。如图 3.4 所示，分别剪断相应的三条 Voronoi 边，得到剪断点 p_{11} 与 p_{12}、p_{21} 与 p_{22}、p_{31} 与 p_{32}。如果适当的选 h 条 Voronoi 边进行剪断，根据引理 3.3，多边形的 Voronoi 图变成了一棵树。

令一个 Voronoi 顶点为树的根结点，其他 Voronoi 顶点为非叶结点，修改后的多边形的顶点和所有剪断点为叶结点，且 Voronoi 边为树的边。这种情况下，该树有 $n+k+2h$ 个叶结点、m 个非叶结点、$e+h$ 条边。

由于每个 Voronoi 顶点的度至少是 3[Held1991]，所以树的根结点至少有 3 个儿子，非叶结点至少有 2 个儿子。根据引理 3.1，得到

$$m \leqslant n+k-2+2h$$

证毕。

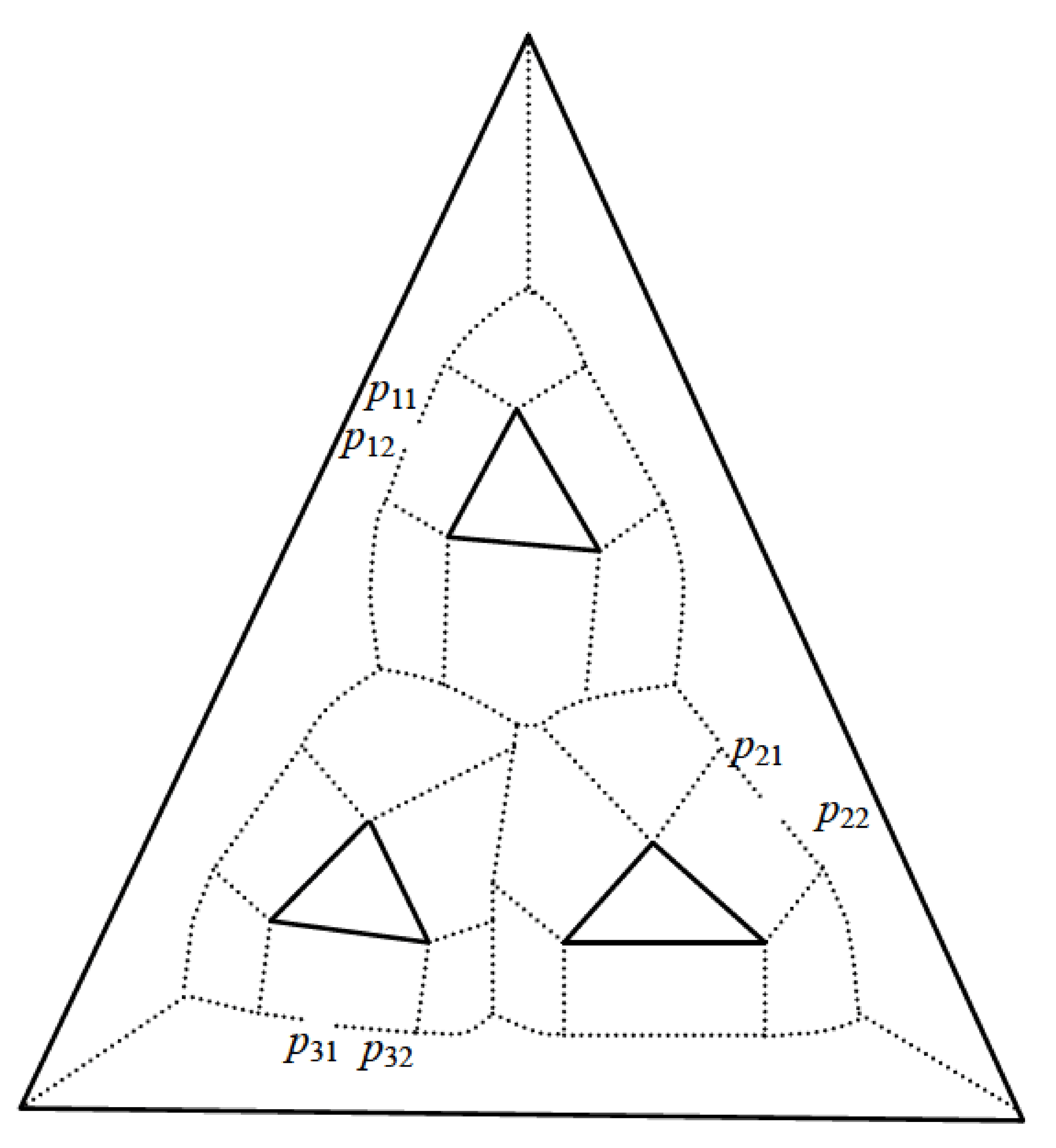


图 3.4　内部 Voronoi 图和被剪断的边

定理 3.2　对于多边形的内部 Voronoi 图，有：

$$e \leqslant 2(n+k)+3h-3$$

证明：根据定理 3.1 的证明，修改后的多边形的内部 Voronoi 图形成一棵树，其共有 $n+k+2h$ 个叶结点、m 个非叶结点和 $e+h$ 条边。由于一棵树的结点数比边数大 1，所以，有

$$e+h \leqslant (m+n+k+2h)-1$$

$$e \leqslant m+n+k+h-1$$

根据定理 3.1，有

$$e \leqslant (n+k-2+2h)+n+k+h-1$$

$$e \leqslant 2(n+k)+3h-3$$

证毕。

推论 3.1　对于多边形 P 的内部 Voronoi 图的每个 Voronoi 区域，有

（1）包含 Voronoi 边的平均数 a_e 小于 5；

（2）包含 Voronoi 顶点的平均数 a_v 小于 4。

证明：在多边形的内部 Voronoi 图中，对于一个边站点 o，必然存在一个 VR(o)；对于一个顶点站点 o，只有 o 是内尖点时才存在一个 VR(o)。因此，VD(P)有 $n + k$ 个 Voronoi 区域，且两个相邻的 Voronoi 区域共享一条 Voronoi 边。我们得到，$(n + k)\, a_e = 2e$。

根据定理 3.2，有

$$(n + k)a_e \leqslant 2(2(n + k) + 3h - 3) \leqslant 4n + 4k + 6h - 6$$

$$a_e \leqslant 4 + 6h / (n + k) - 6 / (n + k)$$

在多边形中，每一条内边界的顶点数大于或等于 3，且内尖点数也大于或等于 3。所以，有 $n + k > 6h$。

我们得到，$a_e < 4 + (n + k) / (n + k) - 6 / (n + k) < 5 - 6 / (n + k)$。

令 a_e 是一个整数，我们得到 $a_e<5$。

在一个 Voronoi 区域中，Voronoi 顶点数等于 Voronoi 边数减 1。我们得到 $a_v<4$。

证毕。

定理 3.3　对于单边界多边形 P 的内部 Voronoi 图，有：

(1)　$m \leqslant n+k-2$

(2)　$e \leqslant 2(n+k)-3$

推论 3.2　对于单边界多边形 P 的内部 Voronoi 图的每个 Voronoi 区域，有：

（1）包含 Voronoi 边的平均数 a_e 小于 4；

（2）包含 Voronoi 顶点的平均数 a_v 小于 3。

证明：根据定理 3.3，有：

$$(n+k)\,a_e = 2e \leqslant 2(2(n+k)-3) \leqslant 4n+4k-6$$

$$a_e \leqslant 4-6/(n+k)$$

得 $a_e < 4$，且 $a_v < 3$。证毕。

引理 3.4 令 j、i 分别是一棵树中的非叶结点和叶结点的数目。如果每个非叶结点至少有 2 个儿子，则：

$$j = i - 1 - \Sigma(\text{每个非叶结点儿子数目} - 2)$$

如果考虑每一个 Voronoi 顶点的度，根据引理 3.4 可以得到如定理 3.4 所示的精确公式。

定理 3.4 对于多边形的内部 Voronoi 图，有：

（1）$m \leqslant n+k-1+2h-\Sigma$（每个非叶结点儿子数目$-2$）

（2）$e \leqslant 2(n-k)-2+3h-\Sigma$（每个非叶结点儿子数目$-2$）

根据引理 3.4，采用类似定理 3.1 和定理 3.2 的方法很容易证明定理 3.4。

2. 多边形的外部 Voronoi 图的性质

由第 2 章我们知道，对于点集 P 的 Voronoi 图，点 p 是点集 P 凸包上的点，当且仅当 p 的 Voronoi 区域是非封闭的。对于多边形的外部 Voronoi 图，也有类似的性质。

引理 3.5 多边形凸包上的顶点必是多边形的凸顶点。

定理 3.5 对于多边形 P 的站点 p，$VR(p)$是非封闭的当且仅当 p 在 P 的凸包上。

证明：

（1）**必要性**：如图 3.5（a）所示，令 $p=AB$（或 C）分别是 P 的凸包上

的站点。则可通过 AB（或 C）作直线 EF（或 GH），使 P 的顶点和边都在 EF（或 GH）的同侧。过 AB 上的任意一点 M（或 C）作 EF（或 GH）的垂线 MS_2（或 CS_1）。显然，从 MS_2（或 CS_1）上的任意一点 S_2（或 S_1）到 P 上任一其他站点的距离都大于 $d(S_2, M)$（或 $d(S_1, C)$）。因此，S_2（或 S_1）在 VR(AB)（或 VR(C)）中。由于 S_2（或 S_1）可以是射线 MS_2（或 CS_1）上的任意一点，所以 VR(AB)（或 VR(C)）是非封闭的。

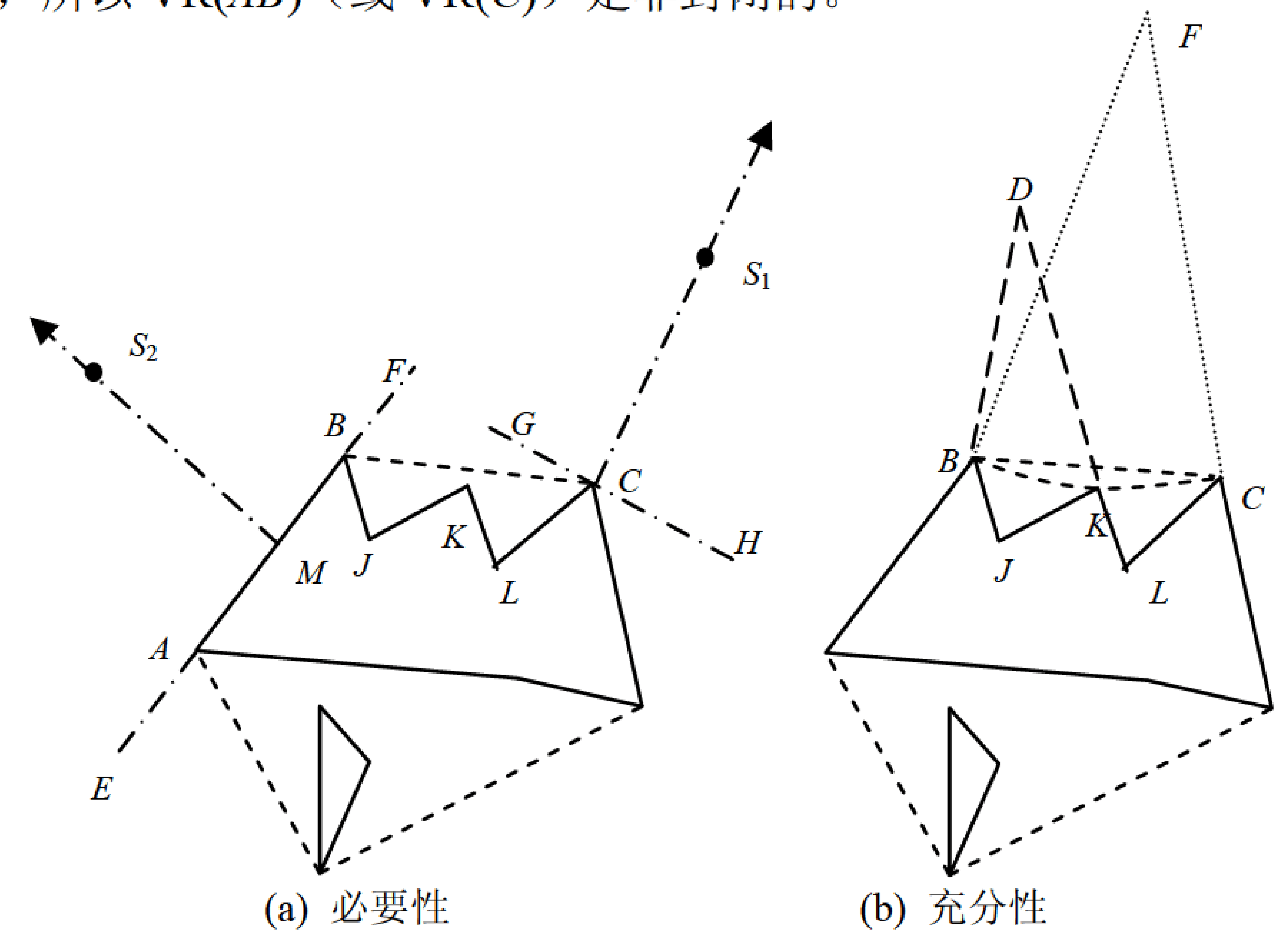


图 3.5　定理 3.5 的证明图例

（2）**充分性：**假定 P 的站点 p 不在 P 的凸包上，现证明 VR(p)是封闭的。如图 3.5（b）所示，设给定站点 p 是边 JK（或顶点 K），可找到 P 的凸包上边 BC，且 P 的边界上 B 到 C 的链中包含边 JK（或 K），JK（或 K）在 CB 的左侧。令 L_1 表示 P 的边界上 B 到 C 的链。过 B 和 C 作一个圆弧，该圆弧除了与 L_1 的一些站点相切外不和它们相交。F 是圆心，可知该圆弧的半径是有限的。易证 P 的外部及三角形△FBC 的外部任一点（例如 D），其到 B 或 C 的距离小于 D 到 L_1 上的任意站点 p 的距离，也就是 VR(p)的任意一点必然落在 CFB 和 L_1 围成的多边形内。即 VR(p)是有界的，也是封闭的。证毕。

由引理 3.5 和定理 3.5，可以得到：$r \leqslant t \leqslant s \leqslant n$，且 $r \geqslant 0$，$t \geqslant 3$，$s \geqslant 3$。

定理 3.6 对于多边形的外部 Voronoi 图，有

$$m \leqslant n+s+2h-r-t-2$$

证明： 作一个圆 C，使多边形 P 的所有顶点和 VD(P)的所有顶点都包含在它内部（见图 3.6）。根据定理 3.5，只要这个圆足够大，C 只与 VD(P)的所有非封闭 Voronoi 区域相交，并被分成 $t+r$ 条弧。由 C 上的 $t+r$ 条弧和 VD(P)组成的图 $G=(V,E)$表示如下：

$V=V_1 \cup V_2 \cup V_3$，

$E=E_1 \cup E_2$。

其中，

$V_1=$ {Voronoi 顶点}，

$V_2=$ {C 与 Voronoi 边的交点}，

$V_3=$ {多边形 P 的顶点}，

$E_1=$ {Voronoi 边}，

$E_2=$ {C 被所有非封闭 Voronoi 区域分成的 $t+r$ 条弧}，

$|V|=|V_1|+|V_2|+|V_3|=m+t+r+n$，

$|E|=|E_1|+|E_2|=e+r+t$。

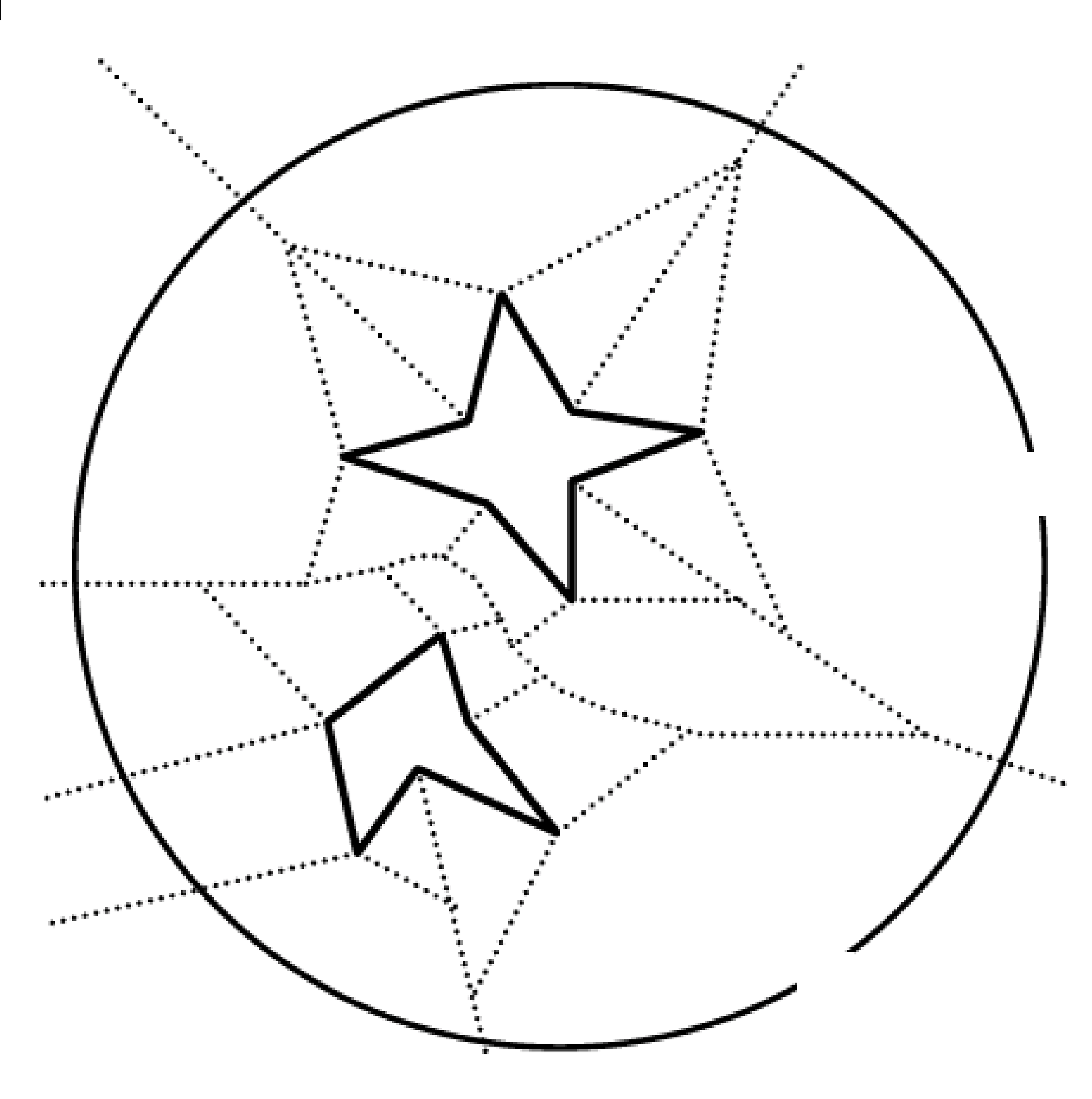


图 3.6 定理 3.6 的证明图例

但是，图 $G=(V,E)$具有 $s+h$ 个环，其中包括 s 个凸顶点所在的环和包围边界的 h 个环。为了得到一棵树，我们做如下处理。

首先，将 V_3 中的 P 的每个凸顶点分成 P 的两个顶点，并在两顶点之间连线，作为 P 的一条边。则有：

$G'=(V',E)$，

$V'=V_1\cup V_2\cup V_3'$，

$V_3'=V_3\cup\{P$ 的每个凸顶点新分成的一个顶点$\}$，

$|V'|=|V_1|+|V_2|+|V_3'|=m+t+r+n+s$，

$|E|=e+r+t$。

然后，在每一个环上，我们在 C 或 VD(P)上适当的选择 h 条圆弧或 Voronoi 边，分别在中点处把它们剪断，这些中点分别变成 2 个剪断点，圆弧或 Voronoi 边由一条变成了两条（见图 3.6）。我们得到：

$G''=(V'',E')$，

$V''=V_1\cup V_2\cup V_3'\cup V_4$，

$E_1'=E_1'\cup E_2'$，

$V_4=\{$新生成的剪断点$\}$，

$E_1'=\{$未被剪断的 Voronoi 边$\}\cup\{$由被剪断的 Voronoi 边生成的边$\}$，

$E_2'=\{$未被剪断的圆弧$\}\cup\{$由被剪断的圆弧生成的边$\}$，

$|V''|=|V_1|+|V_2|+|V_3'|+|V_4|=m+t+r+n+s+2h$，

$|E'|=e+r+t+h$。

在图 $G''=(V'',E')$中取一个度为 3 的非叶结点作为根结点，这样，就得

到一棵树。该树共有 $e+r+t+h$ 条边、$m+r+t$ 个非叶结点、$n+s+2h$ 个叶结点。根据引理 3.1，有：

$$m+r+t \leqslant n+s+2h-2$$

$$m \leqslant n+s+2h-r-t-2$$

证毕。

定理 3.7 对于多边形的外部 Voronoi 图，有

$$e \leqslant 2n+2s+3h-r-t-3$$

证明：由于一棵树的结点数比边数多 1 个。根据定理 3.6 的证明，有：

$$e+r+t+h+1=n+s+2h+m+r+t$$

$$e=n+s+m+h-1$$

根据定理 3.6，有：

$$e \leqslant 2n+2s+3h-r-t-3$$

证毕。

推论 3.3 对于多边形的外部 Voronoi 图 VD(P)中的每一个 Voronoi 区域，有：

（1）其边界上的 Voronoi 边的平均数小于 4;

（2）其边界上的 Voronoi 顶点的平均数小于 3。

证明：令 a_e 和 a_v 分别表示多边形 P 的外部 Voronoi 图 VD(P)的一个 Voronoi 区域的边界上的 Voronoi 边和顶点的平均数。由于 VD(P)有 $n+s$ 个 Voronoi 区域，且两个相邻的 Voronoi 区域共有一条 Voronoi 边，所以有

$$a_e(n+s)=2e \leqslant 2(2n+2s+3h-r-t-3) \leqslant 4(n+s)+6h-2(r+t)-6$$

$$a_e \leqslant 4+6h/(n+s)-2(r+t)/(n+s)-6/(n+s)$$

$$a_e < 5 - 2(r + t) / (n + s) - 6 / (n + s)$$

由于 a_e 是一个整数，所以，$a_e < 5$。

由于在一个 Voronoi 区域的边界上，有 $a_v < a_e - 1$，所以，$a_v < 4$。

证毕。

3. 凸多边形的外部 Voronoi 图的性质

凸多边形是一个单边界多边形，只有一条封闭的边界。对于凸多边形，由于其顶点和边都在其凸包上，即有 $r = t = s = n$，且 $h = 1$，所以它的外部 Voronoi 图的性质存在一定的特殊性（如图 3.7 所示）。

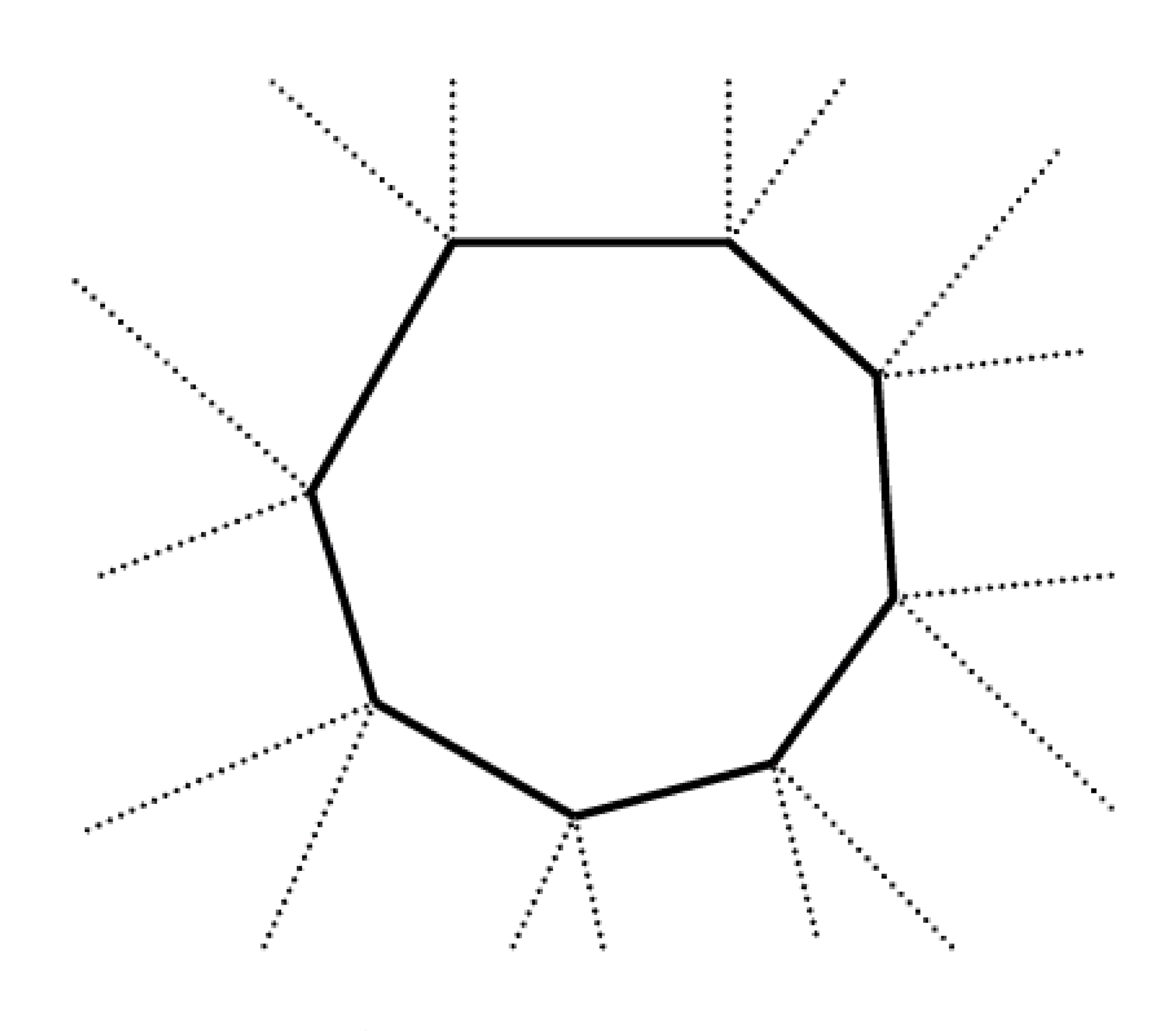


图 3.7　凸多边形（实线）及其外部 Voronoi 图（虚线）

推论 3.4　对于凸多边形的每个站点的外部 Voronoi 区域，有：

（1）是非封闭的；

（2）只有 2 条 Voronoi 边；

（3）有 0 个 Voronoi 顶点。

推论 3.5　对于凸多边形的外部 Voronoi 图，有：

（1） $m = 0$；

（2） $e = 2n$。

推论 3.4 和推论 3.5 是比较容易证明的，此处略。

3.2 构造方法

Voronoi 图的计算方法有许多，主要有分而治之算法、扫描线算法、增量算法等。

分而治之算法主要是将站点划分为两部分，递归计算每一部分的 Voronoi 图，再将它们合并，最终得到整个站点集合的 Voronoi 图。文献[Kirkpatrick1979，Lee1982，Yap1987]分别给出了在 $O(n\log n)$时间内计算 PSLG 的 Voronoi 图的分而治之算法。对于直线段为站点的情况，分割生成两个站点子集比点为站点的情况复杂得多，而且连接两个站点子集的 Voronoi 图的步骤也是比较困难且难以实现的。

扫描线算法主要使用扫描线在平面上逐渐扫描生成站点，并不断更新 Voronoi 图的信息，最后得到整个站点集合的 Voronoi 图。扫描线算法可以看作一种特殊的增量算法。文献[Fortune1986]给出了可在 $O(n\log n)$时间内构造直线段 Voronoi 图的扫描线算法。

增量算法通过每次添加一个生成站点并不断在局部范围内修改已有 Voronoi 图来获取新 Voronoi 图的方式来计算整个站点集合的 Voronoi 图。文献[Boissonnat1992，Klein1993，Held2001]给出了在 $O(n\log n)$时间内构造 PSLG 的 Voronoi 图的增量算法。文献[Held2001]以文献[Sugihara2000]中的拓扑向导（Topology-Oriented）方法为基础，实现了一种计算点和线段的 Voronoi 图的增量算法。该算法从处理站点的空集开始，Voronoi 图也相应为空；然后每次向已处理的站点集合中添加一个新的站点，并对已有 Voronoi 图做相应修改，最终得到整个站点集合的 Voronoi 图。每次对已有 Voronoi 图做相应修改生成新的 Voronoi 图时，依据 Voronoi 图本身固有的拓扑属性，删除原来的 Voronoi 顶点，增加新的 Voronoi 图顶点。

文献[Aggarwal1989]给出了计算凸多边形的 Voronoi 图的 $O(n)$算法。文献[Klein1995，Chin1995，Chin1998]则给出了计算单边界多边形的 Voronoi 图的 $O(n)$算法。文献[Chin1995]首先将简单多边形分解为正则直方图

（Pseudonormal Histograms），然后通过分解为影响区域直方图（Influence Histograms），最后转化为 *xy* 单调的直方图（*xy* Monotone Histograms）；计算 *xy* 单调的直方图的 Voronoi 图，将其合并得到整个多边形的 Voronoi 图。相关的详细综述信息可阅文献[Sack2000，Held2001]等。

下面给出一种比较简单的多边形 Voronoi 图构造算法，该算法虽然复杂度较高，但易于理解与实现。

算法采用半平面法构造多边形的 Voronoi 图，为多边形每个站点独立地计算其 Voronoi 区域，即计算其 Voronoi 区域边界的各条 Voronoi 边。

算法 3.1　半平面法构造多边形的 Voronoi 图

（1）为多边形各站点编号，计算两两站点间的平分线；

（2）为多边形每个站点计算其 Voronoi 区域：

① 计算当前站点与其前后相邻站点的平分线；

② 设定当前站点与其前一个相邻站点的平分线为当前平分线；

③ 计算当前站点其他相关平分线与当前平分线的有效交点；

④ 按照与当前平分线起始点的距离大小对这些交点进行排序；

⑤ 将最近交点作为当前平分线的终点，输出一条 Voronoi 边；

⑥ 选择形成最近交点的平分线作为当前平分线；

⑦ 若当前平分线为当前站点与后一个相邻站点的平分线，转到⑧，否则转③；

⑧ 完成对该站点 Voronoi 区域的计算；

（3）输出多边形的 Voronoi 图。

3.3 应用实例

本节主要介绍多边形的 Voronoi 图的应用，包括凸多边形间的求交与距离计算，可见性计算以及路径规划等，并介绍这些算法在虚拟博物馆中的集成应用。

3.3.1 两个凸多边形的求交计算

凸多边形间的求交计算在计算机游戏、计算机动画、虚拟现实、运动规划、图形学等方面应用广泛。目前已有很多算法，最著名的是 O'Rourke 的追逐算法[O'Rourke1994]，其时间复杂度为 $O(n)$。这里，我们介绍一种基于外部 Voronoi 图的凸多边形间的求交算法，时间复杂度仍为 $O(n)$，但算法效率和鲁棒性更好[YangCL2006a]。

由 3.2 节可知，凸多边形的外部 Voronoi 图非常特殊，每条 Voronoi 边都垂直于凸多边形的一条边。显然，对于一个凸多边形来说，在逆时针方向上顶点和边的 Voronoi 图是交替出现的，如图 3.7 所示。

算法的主要思想是：让两凸多边形 P 和 Q 的并集和交集的边界上的两边 a 和 b 相互追逐，并计算它们的交点，直到找到了两多边形的所有交点，或者是成功地判定了两多边形的空间位置（包含或分离）。在追逐过程中，b 沿着一个多边形（不妨假设为 Q，这时遍历的 P 和 Q 的并集的边界在 Q 上）的边界连续移动并穿过另一个多边形（不妨假设为 P）的外部 Voronoi 区域，a 同时在 P 的边界上相应地移动，直到 a 和 b 相交于一点。这时，a 和 b 互换，b 沿着 P 的边界移动并穿过 Q 的外部 Voronoi 区域，a 同时在 Q 的边界相应地移动。因此，b 总是在 a 所在多边形的外部。

算法的主要步骤是判断 b 和另一个凸多边形的外部 Voronoi 图的关系，据此决定是沿着原来的多边形的边界继续遍历，还是更换到另一个多边形的边界上进行遍历。如图 3.8 所示，a 是 P 的边 p_jp_{j+1}；b 是 Q 的边，s 是 b 的起点，也是 Q 的顶点，其位于 VR(a)内。b 与 VR(a)存在以下三种可能的关系。

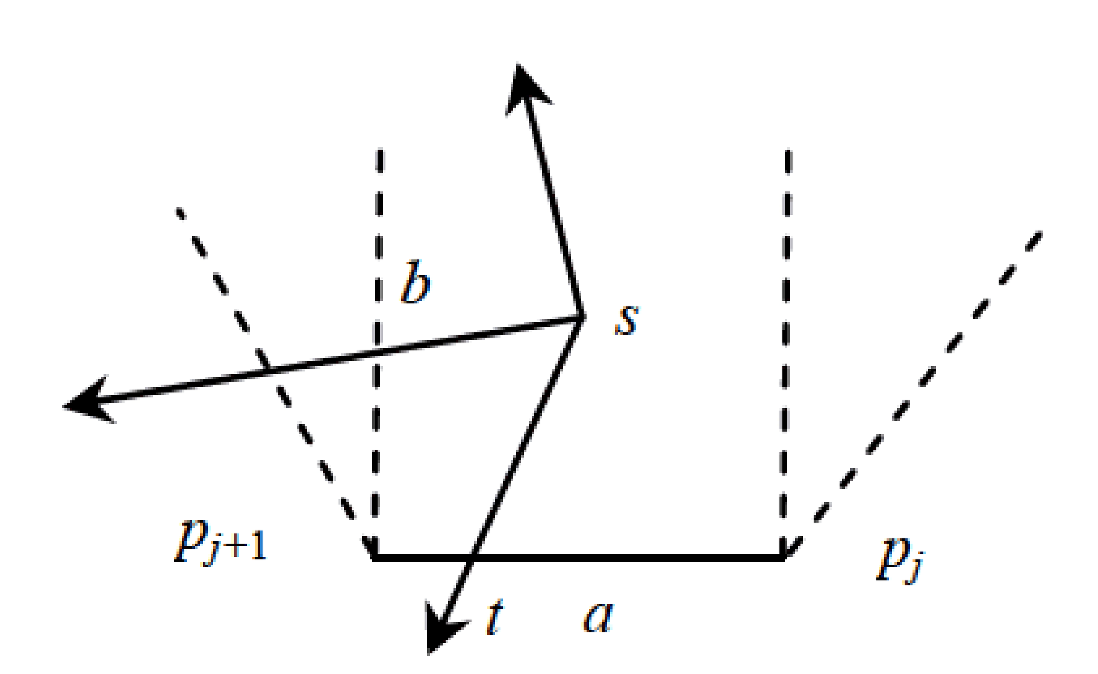


图 3.8　被遍历的边与另一多边形的外部 Voronoi 图的关系

- b 与 VR(a)的边界不相交：这时，令 s 为 b 的另一端点，即令 b 在 Q 上前进一条边，继续新的遍历。
- b 与 VR(a)的 Voronoi 边相交：这时，由于 b 首先穿过 VR(p_{j+1})的边界，取 $a=p_{j+1}$，继续新的遍历。
- b 与 a 相交：设 a 与 b 相交于点 t，令 $a=b$，$b=t\,p_{j+1}$，$s=t$，继续新的遍历。这里，t 为找到的 P 与 Q 的一个交点。

通过上述三种情况的处理，保证了 b 总是在 $P\cup Q$ 的边界上遍历，a 总是在 $P\cap Q$ 的边界上遍历，a 和 b 一旦相遇，即可得到 P 与 Q 的一个交点，这时需要交换 a 和 b 后继续遍历。

算法 3.2　两个凸多边形的求交计算

（1）找出两凸多边形的所有顶点中的最低的顶点（如果存在多个，取最左侧的那个），将其作为整个遍历的起点，记为 s_0。令 $s=s_0$。用 Q 表示 s_0 所在的凸多边形，P 表示另一个凸多边形。

（2）判断 s 所在的 P 的 Voronoi 区域。设 s 在 P 的站点 $p_j p_{j+1}$（顶点可以看做特殊的边）的 Voronoi 区域中，令 b 为以 s 为起点的 Q 的边（逆时针方向），$a=p_j p_{j+1}$。

（3）令 $L=\{\}$。//用于记录 P 与 Q 的交点。

（4）如果 b 与 VR(a)的边界不相交，则令 s 为 b 的另一端点，b 为 s 起点的 Q 的边（逆时针方向），转（7）。

（5）如果 b 与 VR(a)的 Voronoi 边相交，则 $a = p_{j+1}$，$j = j + 1$，转（7）。

（6）如果 b 与 a 相交，则计算 a 与 b 的交点 t，令 $a = b$，$b = t\,p_{j+1}$，$p_j = s$，$s = t$，L=$L \cup \{t\}$，转（7）。

（7）如果 $s == s_0$，转（8）；否则，转（4）。

（8）如果 L!={}，则 L 中的点为 P 与 Q 的所有交点，依次输出它们。

（9）否则，P 与 Q 没有交点。

上述算法最后可以给出 P 与 Q 是否相交的结果。如果它们不相交，可进一步判断它们的空间位置关系，即是分离还是包含关系：如果整个算法过程中，a 和 b 未发生交换，且只访问了 P 的部分顶点，则 P 与 Q 是分离的，否则 Q 包含 P。

另外，对上述算法稍作修改，可让 L 记录 P 与 Q 交集的所有顶点。类似地，也可以记录 P 与 Q 并集的所有顶点。

3.3.2　两个分离凸多边形的距离计算

本节主要讨论两个分离凸多边形间的距离计算问题。目前解决这个问题的最优算法的时间复杂度为 $O(\log m + \log n)$ 。本节对两个分离凸多边形的外部 Voronoi 图之间的关系进行深入分析，并介绍基于 Voronoi 图来计算两凸多边形间距离的 $O(\log m + \log n)$ 算法[YangCL2006b]。

1. 算法思想

用 P 和 Q 表示欧氏几何空间中的两个分离的凸多边形。P 和 Q 之间的距离为：

$d(P, Q)$=min{$d(p, q)$|p，q 分别是 P 和 Q 的点，$d(p, q)$为 p 与 q 间的欧氏距离}

如果有 P 和 Q 的点 p^*和 q^*，满足 $d(p^*, q^*)=d(P, Q)$，则称<p^*，q^*>为 P 和 Q 的一个最短距离实现点对，向量 $\boldsymbol{v}^*=q^*-p^*$为 P 和 Q 的一个最短距离实

现向量。对于两个分离的凸多边形，其最短距离实现点对不一定是唯一的，但最短距离实现向量是唯一的。由于 P 和 Q 是凸多边形，可知 p^*和 q^*分别在 P 和 Q 的边界上。

分别用 o_p 和 o_q 表示 P 和 Q 边界上的边或顶点，如果满足 $d(o_p, o_q)=d(P, Q)$，则称< o_p，o_q >为 P 和 Q 的一个最短距离站点对。< o_p，o_q >有如下几种类型。

（1）<顶点，顶点>

o_p、o_q 是顶点。其只包含一个最短距离实现点对<p^*, q^*>，其中 $p^*=o_p$，$q^*=o_q$；

（2）<顶点，边>

o_p 是顶点，o_q 是边。其只包含一个最短距离实现点对<p^*, q^*>，其中 $p^*=o_p$，$q^*=p(o_p, o_q)$，$p(o_p, o_q)$为 o_p 在边 o_q 上的投影（垂心）。此时，q^*必在线段 o_q 上，且一定有 $q^*\in \mathrm{VR}(o_p)$。

类似地，可以 o_p 是边，o_q 是顶点。

（3）<边，边>

o_p，o_q 都是边，且 o_p 与 o_q 平行。这时，存在多个最短距离实现点对(但最短距离实现向量是唯一的)。这种情况下，其最短距离实现点必存在于 o_p，o_q 及其顶点形成的<顶点，顶点>或<顶点，边>情况中。因此，我们只考虑对类型（1）和（2）进行处理即可。

通过上述分析知，如果已知 P 和 Q 之间的最短距离站点对，可根据其类型在 $O(1)$时间内计算出 p^*，q^*以及 P，Q 之间的最短距离：$d(P, Q)=d(p^*, q^*)$。

定理 3.8　对于任意两个站点 $o_p\in P$ 和 $o_q\in Q$，p^*和 q^*为表示 o_p 和 o_q 之间最短距离的一对点。o_p 和 o_q 是 P 和 Q 的一个最短距离站点对，当且仅当 $p^*\in \mathrm{VR}(o_q)$且 $q^*\in \mathrm{VR}(o_p)$[Lin1991]。

证明：分别过 p^*和 q^*作线段 p^*q^*的垂线 l_1 和 l_2（如图 3.9、图 3.10 所示）。

易证 p^*和 q^*是 P 和 Q 之间最短距离的一对点的充要条件为 P 在垂线 l_1 的左侧，且 Q 在垂线 l_2 的右侧。而 P 在垂线 l_1 左侧及 Q 在垂线 l_2 右侧的充要条件为线段 p^*q^*属于 VR(oq)及 VR(op)。证毕。

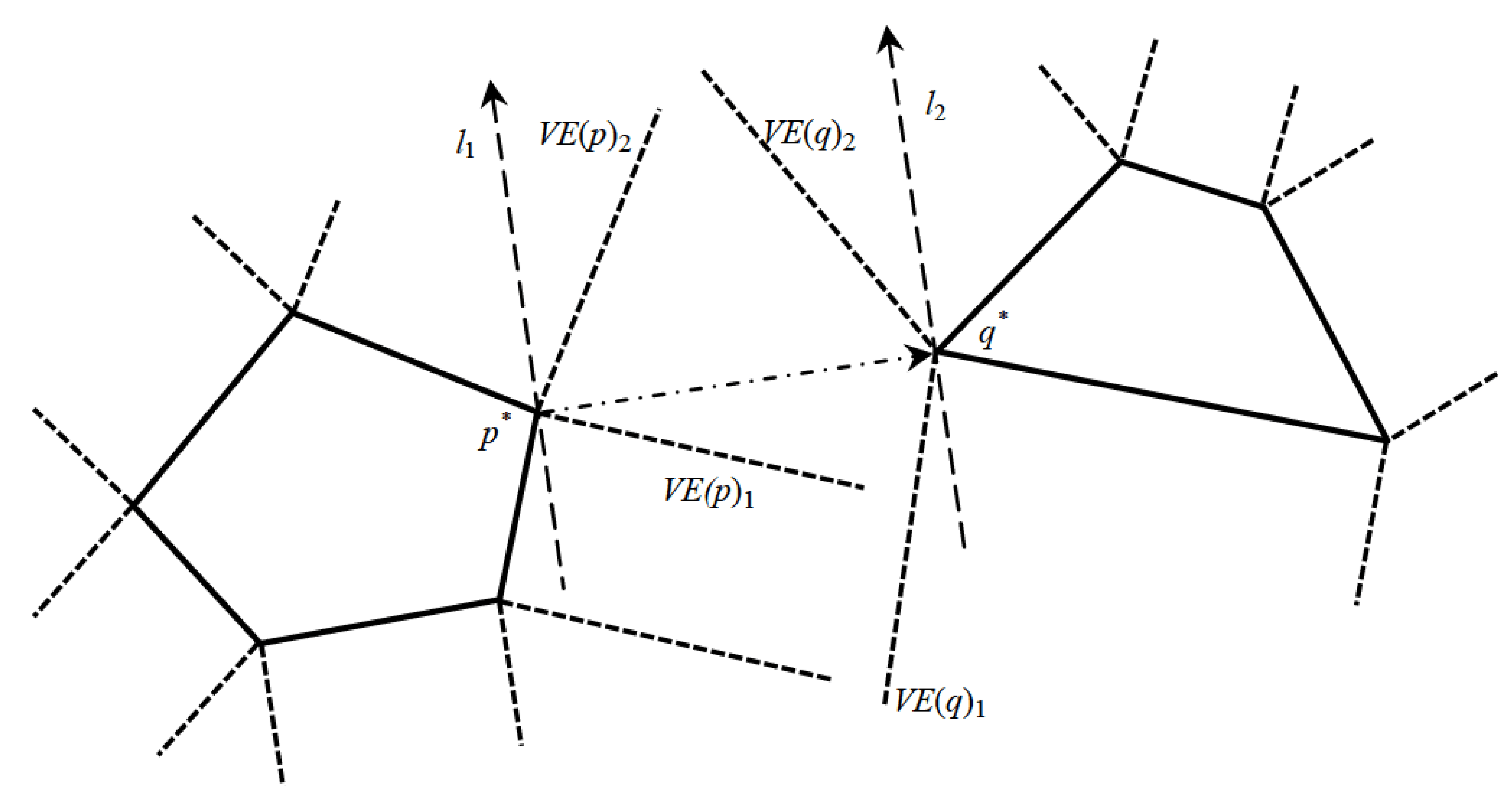


图 3.9　最短距离站点对：<顶点，顶点>

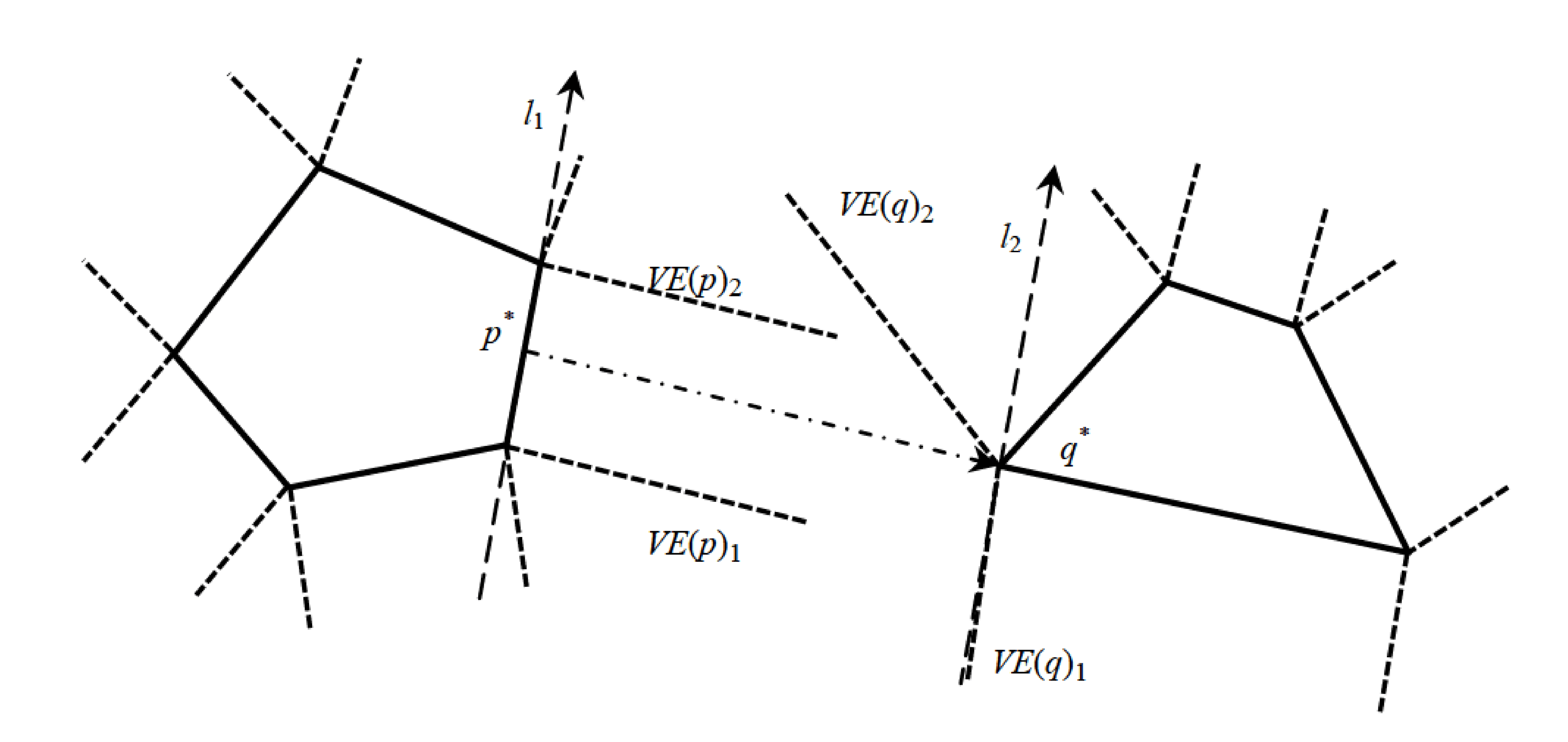


图 3.10　最短距离站点对：<顶点，边>

根据定理 3.8 知，对于分别属于两个凸多边形的任意两个站点，如果表示它们之间的最短距离的一对点中的每个点属于另一个站点的 Voronoi 区域，则这两个站点之间的最短距离就是两个凸多边形之间的最短距离。

为了计算P和Q之间最短距离站点对$<o_p, o_q>$，我们首先采用二分法查找包含o_p，o_q的初始搜索范围，然后仍采用二分法逐渐缩小搜索范围，直到查找到$<o_p, o_q>$。我们用C(p_1，p_2)表示多边形边界上从顶点p_1沿逆时针方向到顶点p_2的多边形链。结合图3.11，算法思想简单叙述如下。

（1）首先，在P和Q中确定两个初始搜索范围P'=C(p'，p'')，Q'=(q''，q')，这里要保证P'和Q'包含P和Q的一个最短距离站点对$< o_p, o_q >$(后面会具体介绍如何保证这一点)。

（2）然后，分别取P'、Q'中间的站点p_a、q_b，它们将P'、Q'分成P_1''=C(p'，p_a)和P_2''=C(p_a，p'')，Q_1''=C(q''，q_b)和Q_2''=C(q_b，q')。

（3）再根据p_a、q_b、VR(p_a)和VR(q_b)的位置关系，确定P_1''、P_2''、Q_1''和Q_2''中有一个或两个肯定不包含P和Q的最短距离站点对。将肯定不包含P和Q的最短距离站点对的多边形链删除，得到新的搜索范围。例如，如果只确定删除P_1''，则新的搜索范围为P_2''和Q'；如果确定删除P_1''和Q_1''，则新的搜索范围为P_2''和Q_2''。

（4）重复上面的搜索过程，直到最终得到最短距离站点对$<o_p, o_q>$。最后，根据$<o_p, o_q>$的类型，计算P和Q的距离。

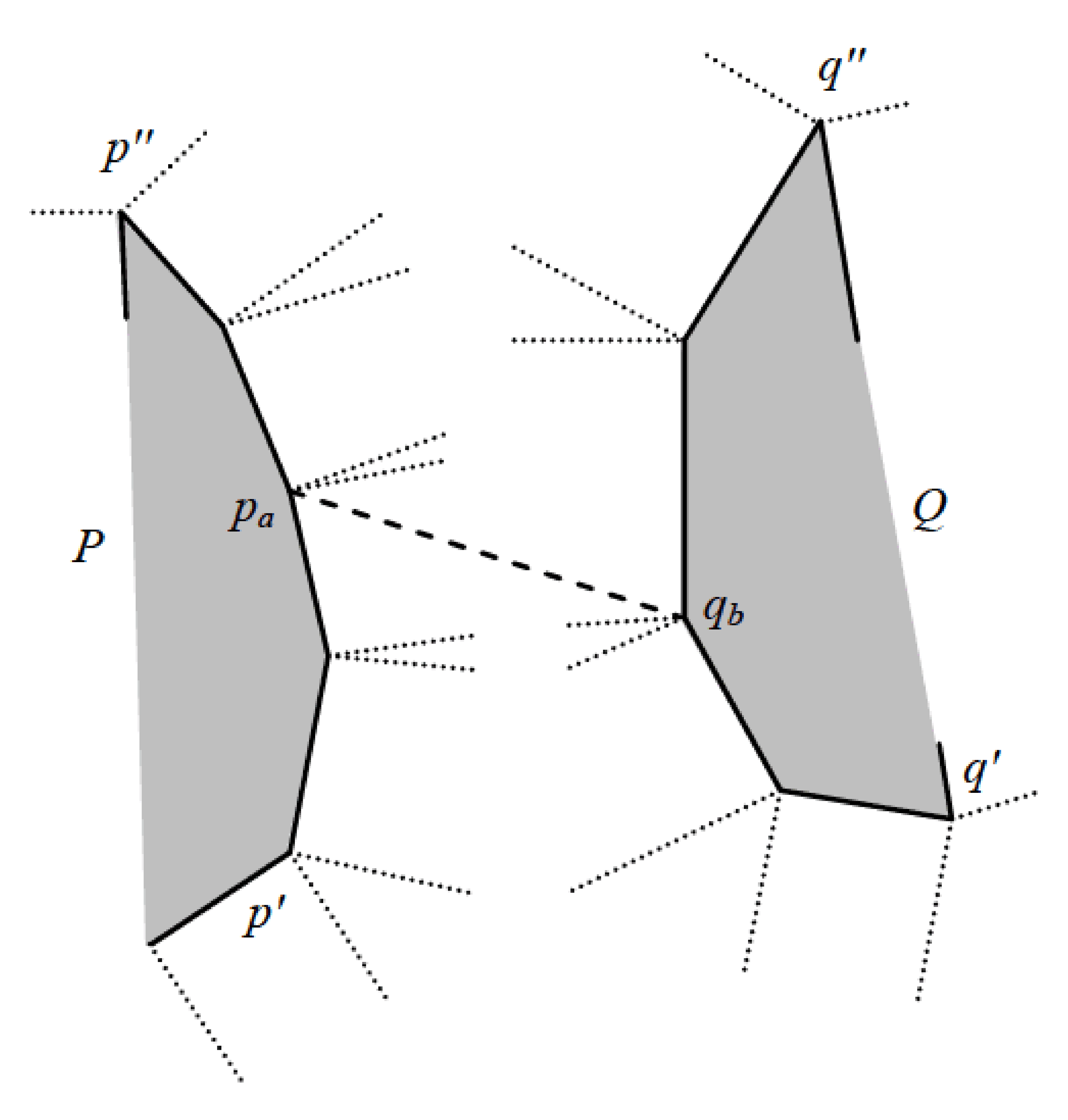


图 3.11　用二分法计算最短距离站点对

算法要在 $O(\log n+\log m)$时间内找到最短距离站点对，须解决以下两个关键问题。

（1）在 $O(\log n+\log m)$时间内计算出初始搜索范围 P'和 Q'。其中 P'和 Q'应包含最短距离站点对<o_p，o_q>，且 P'和 Q'包含的站点越少越好。

（2）在 $O(1)$时间内根据 p_a、q_b、VR(p_a)和 VR(q_b)的位置关系，决定删除 P_1''、P_2''、Q_1'' 和 Q_2'' 中的哪一个。

2. 确定初始搜索范围

设 P 的顶点按逆时针方向依次为 p_1，p_2，…，p_n，多边形 Q 的顶点按逆时针依次为 q_1，q_2，…，q_m，其中 n、m 分别为 P 和 Q 的顶点数，如果下标超过 n 或 m，则进行求模运算。

对于一条将 P 和 Q 左、右分开的有向直线，我们称之为 P 和 Q 的一条分离线。对于 P 和 Q 的一个最短距离站点对<o_p，o_q>，p^*q^*表示 o_p，o_q之间距离最短的一对点，则线段 p^*q^*的中垂线必为 P 和 Q 的一条分离线（如图 3.12 所示）。我们称这样的分离线为 P 和 Q 的中分线，其为有向直线，P 和 Q 分别在其左右侧，用 l 表示。这里需要说明的是，本节中中分线 l 只是用于分析两个分离的凸多边形间的位置关系，在算法中并不需要计算。

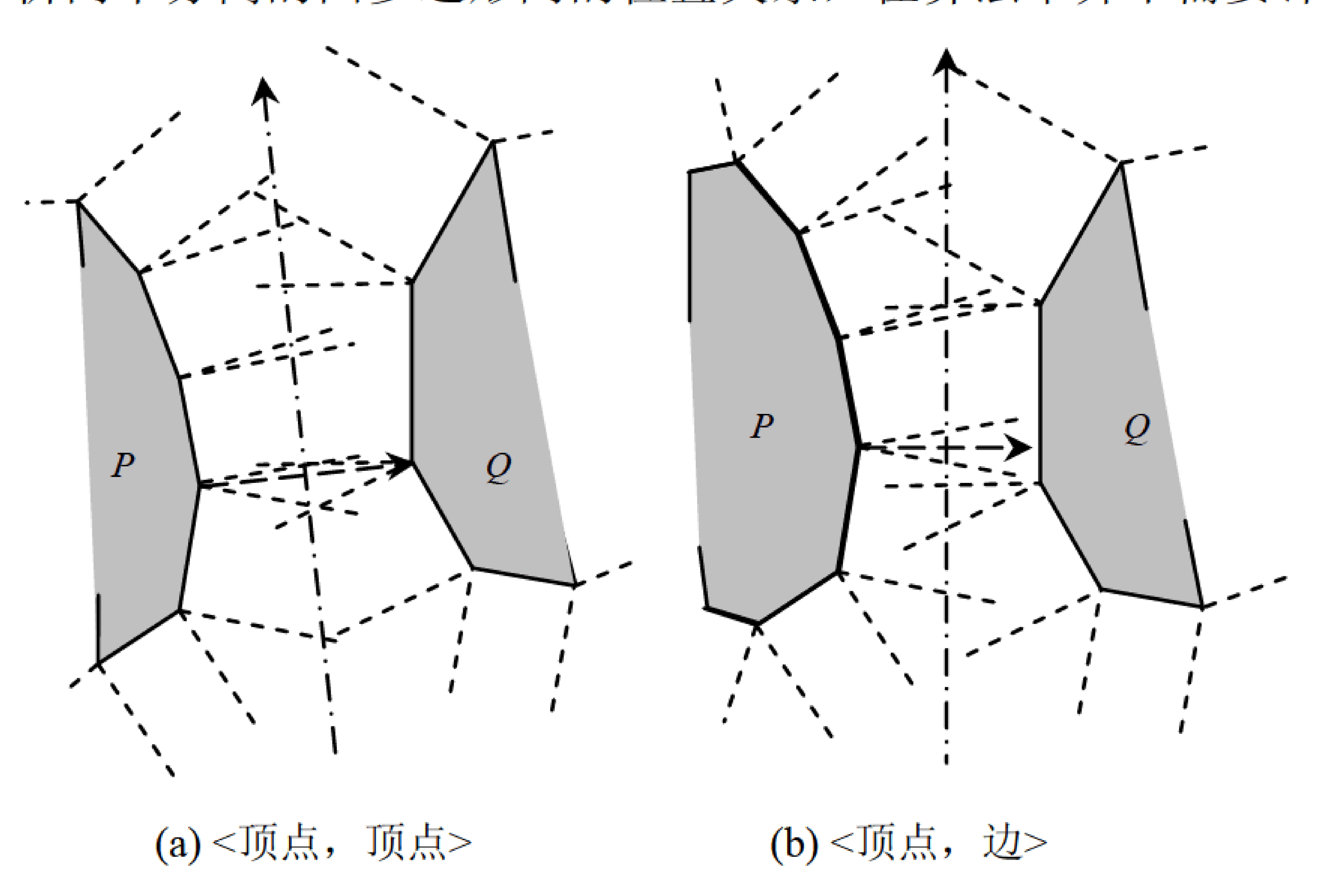


图 3.12　P 和 Q 间的中分线

设 p'、$p''(p'\neq p'')$为多边形 P 上两个顶点，分别过 p'，p''作中分线 l 的垂线 l_1、l_2。如果 P 完全在 l_1、l_2之间，且 l_1 与 P 只交于 p'，l_2 与 P 只交于 p''，则 p'、p''将 P 分成两条多边形链 P_1=C(p'，p'')和 P_2=C(p''，p')。如图 3.13 所示，选择 p'、p''可使 P_1 关于中分线 l 是严格单调的，P_2 关于中分线 l 是单调的[O'Rourke1994]。

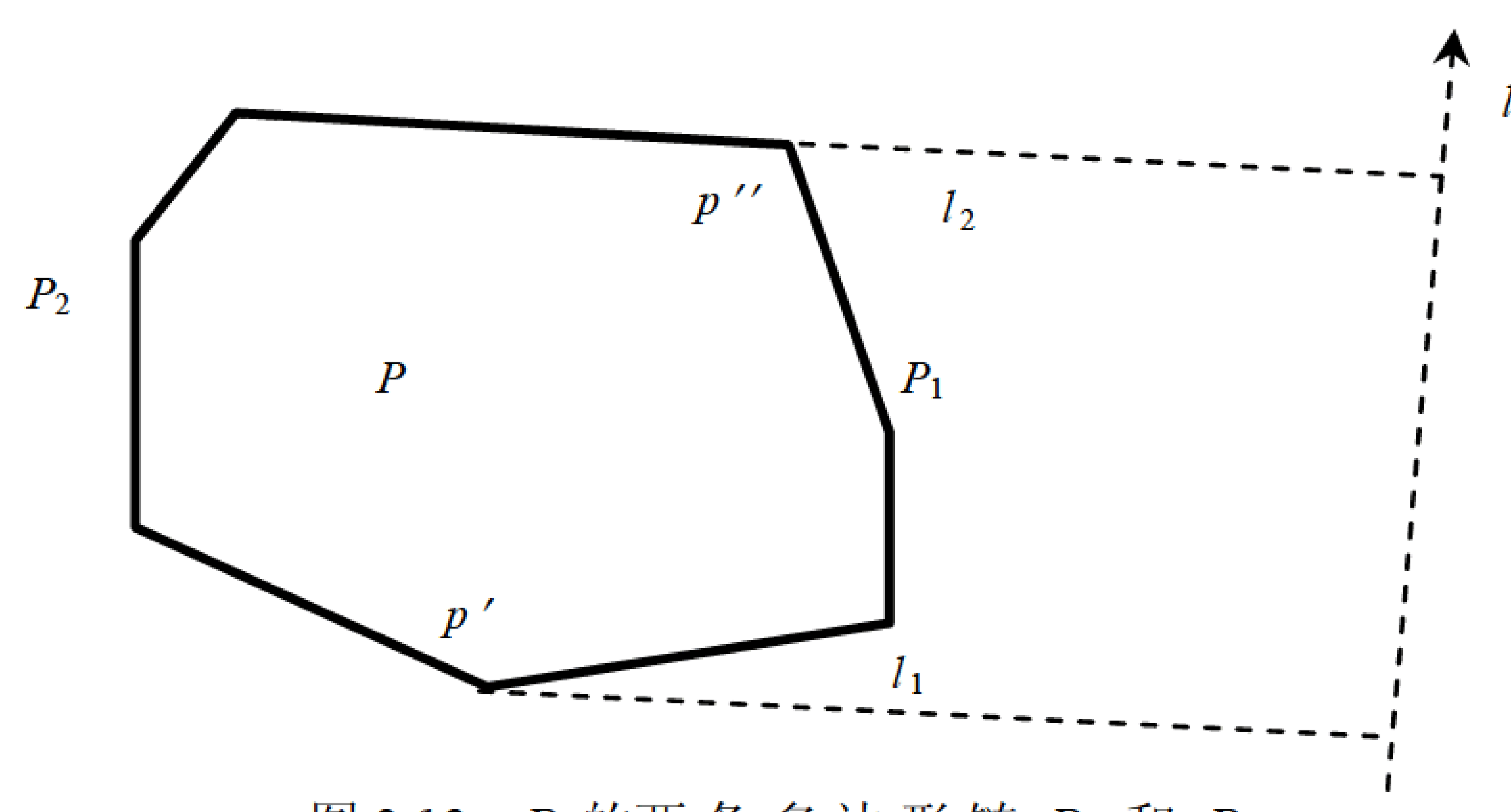


图 3.13　P 的两条多边形链 P_1 和 P_2

对于凸多边形 P 上的任意顶点站点 p_i，用 VE(p_i)$_1$ 和 VE(p_i)$_2$ 分别表示 p_i 的两条 Voronoi 边，其中 VE(p_i)$_1$ 是起点为 p_i 且垂直于 $p_{i-1}p_i$ 的一条射线，$(p_{i-1}p_i\times \mathrm{VE}(p_i)_1)\bullet k<0$；VE($p_i$)$_2$ 是起点为 p_i 且垂直于 p_ip_{i+1} 的一条射线，$(p_ip_{i+1}\times \mathrm{VE}(p_i)_2)\bullet k<0$，其中 k 为 z 轴方向的单位向量。

下面用 l_u，l_{i1}，l_{i2}，l_0，l_1 和 v_u^* 分别表示 l，VE(p_i)$_1$，VE(p_i)$_2$，VE(p')$_2$，VE(p'')$_1$ 和最短距离实现向量的单位向量。

性质 3.1　对于多边形链 P_1 上的任意顶点站点 p_i，有下列性质：

(1) 如果 $p_i=p'$，则 $VE(p_i)_1$ 与 l 不相交，$VE(p_i)_2$ 与 l 相交；

(2) 如果 p_i=p''，则 $VE(p_i)_2$ 与 l 不相交，$VE(p_i)_1$ 与 l 相交；

(3) 如果 $p_i\neq p'$且 $p_i\neq p''$，则 $VE(p_i)_1$、$VE(p_i)_2$ 都和 l 相交，且 $(v_u^*\times l_0)\bullet k<(v_u^*\times l_{i1})\bullet k<(v_u^*\times l_{i2})\bullet k<(v_u^*\times l_1)\bullet k$。

性质 3.2　对于多边形链 P_2 上的任意顶点站点 p_i，有下列性质：

(1) 如果 $p_i=p'$，则 $VE(p_i)_1$ 与 l 不相交，$VE(p_i)_2$ 与 l 相交；

(2) 如果 $p_i=p''$，则 $VE(p_i)_2$ 与 l 不相交，$VE(p_i)_1$ 与 l 相交；

(3) 如果 $p_i \neq p'$ 且 $p_i \neq p''$，则 $VE(p_i)_1$、$VE(p_i)_2$ 都不和 l 相交。

因此，可以证明 P、Q 的最短距离站点对 $<o_p, o_q>$ 中的 o_p 在 P_1 中，其将 P_1 分成三部分：上部分 P_u，中间部分 P_m，下部分 P_d。

性质 3.3 对于多边形链 P_1 上的任意顶点站点 p_i，有下列性质：

(1) $\forall p_i \in P_u$，有 $l_{i1} \bullet l_u > l_{i2} \bullet l_u > 0$，且 $(\mathrm{v}_\mathrm{u}^* \times l_{i2}) \bullet k > (\mathrm{v}_\mathrm{u}^* \times l_{i1}) \bullet k > 0$;

(2) $\forall p_i \in P_d$，有 $l_{i1} \bullet l_u < l_{i2} \bullet l_u < 0$，且 $(\mathrm{v}_\mathrm{u}^* \times l_{i1}) \bullet k < (\mathrm{v}_\mathrm{u}^* \times l_{i2}) \bullet k < 0$;

(3) P_1 的中间部分 P_m 只包含最短距离站点对 $<o_p, o_q>$ 中的站点 o_p，且：

(a) 如果 o_p 是边，则 $l_{i1} \bullet l_u = l_{i2} \bullet l_u = 0$，且 $(\mathrm{v}_\mathrm{u}^* \times l_{i1}) \bullet k = (\mathrm{v}_\mathrm{u}^* \times l_{i2}) \bullet k = 0$;

(b) 如果 o_p 是顶点，则 $l_{i1} \bullet l_u < 0$，$l_{i2} \bullet l_u > 0$，且 $(\mathrm{v}_\mathrm{u}^* \times l_{i1}) \bullet k < 0$，$(\mathrm{v}_\mathrm{u}^* \times l_{i2}) \bullet k > 0$。

类似地，凸多边形 Q 上也可分成两条多边形链 Q_1 和 Q_2，且 Q_1 和 Q_2 仍然保持逆时针方向。Q_1 关于中分线 l 是严格单调的，Q_2 关于中分线 l 是严格单调的或单调的。Q_1、Q_2 中任意站点的 Voronoi 边，也分别具有类似上面所介绍的 P_1、P_2 中站点的 Voronoi 边的性质。P、Q 间的最短距离站点对 $< o_p, o_q >$ 中的 o_q 也将 Q_1 分为三部分：上部分 Q_u，中间部分 Q_m，下部分 Q_d。

对于多边形 Q 上的任意顶点站点 q_j，用 $VE(q_j)_1$ 和 $VE(q_j)_2$ 分别表示 q_j 的两条 Voronoi 边，其中 $\mathrm{VE}(q_j)_1$ 是起点为 q_j 且垂直于 $q_{j-1}q_j$ 的一条射线，$(q_{j-1}p_i \times \mathrm{VE}(q_j)_1) \square k < 0$；$\mathrm{VE}(q_j)_2$ 是起点为 q_j 且垂直于 $q_j q_{j+1}$ 的一条射线，$(q_j q_{j+1} \times \mathrm{VE}(q_j)_2) \square k < 0$。用 l_{j1} 和 l_{j2} 分别表示 $\mathrm{VE}(q_j)_1$ 和 $\mathrm{VE}(q_j)_2$ 方向的单位向量。

性质 3.4 对于多边形链 Q_1 上的任意顶点站点 q_j，有下列性质：

（1） $\forall q_j \in Q_u$，有 $l_{j2} \square l_u > l_{j1} \square l_u > 0$，且 $(\mathrm{v}_\mathrm{u}^* \times l_{j1}) \bullet k > (\mathrm{v}_\mathrm{u}^* \times l_{j2}) \bullet k > 0$;

（2）$\forall q_j \in Q_d$，有 $l_{j2} \bullet l_u < l_{j1} \bullet l_u < 0$，且$(v_u^* \times l_{j2}) \bullet k < (v_u^* \times l_{j1}) \bullet k < 0$；

（3）Q_1 的中间部分 Q_m 只包含最短距离站点对<o_p，o_q>中的站点 q，且：

（a）如果 o_q 是边，则 $l_{j1} \bullet l_u = l_{j2} \bullet l_u = 0$，且$(v_u^* \times l_{j1}) \bullet k = (v_u^* \times l_{j2}) \bullet k = 0$;

（b）如果 o_q 是顶点，则 $l_{j2} \bullet l_u < 0$，$l_{j1} \bullet l_u > 0$，且$(v_u^* \times l_{j2}) \bullet k < 0$，$(v_u^* \times l_{j1}) \bullet k > 0$。

显然，P 和 Q 的最短距离站点对 o_p，o_q 分别位于 P_1，Q_1 中。我们将 P_1，Q_1 中的站点称为候选站点。后面的算法 3.3 可计算出 P 和 Q 的初始搜索范围 P'和 Q'，其中 P'和 Q'分别只是 P_1，Q_1 的一部分，但包含有最短距离站点对。在介绍该算法前，先介绍几个定理。

引理 3.6　对于任意一点 $u \in P$，存在站点 $v \in Q_1$，满足 $u \in \mathrm{VR}(v)$。

引理 3.7　对于任意一点 $u \notin P$，站点 $p' \in P$，$p'' \in P$，$p' \neq p''$，$u \notin \mathrm{VR}(p')$且 $u \notin \mathrm{VR}(p'')$，以及任意顶点 $v \in \mathrm{C}(p', p'')$，$v \neq p'$且 $v \neq p''$。如果 $uv \cap \mathrm{VE}(p')_2 = \varnothing$，且 $uv \cap p'p'' = \varnothing$，且 $uv \cap \mathrm{VE}(p'')_1 = \varnothing$，则存在一个站点 $o \in \mathrm{C}(p', p'')$，满足 $u \in \mathrm{VR}(o)$（如图 3.14 所示）。

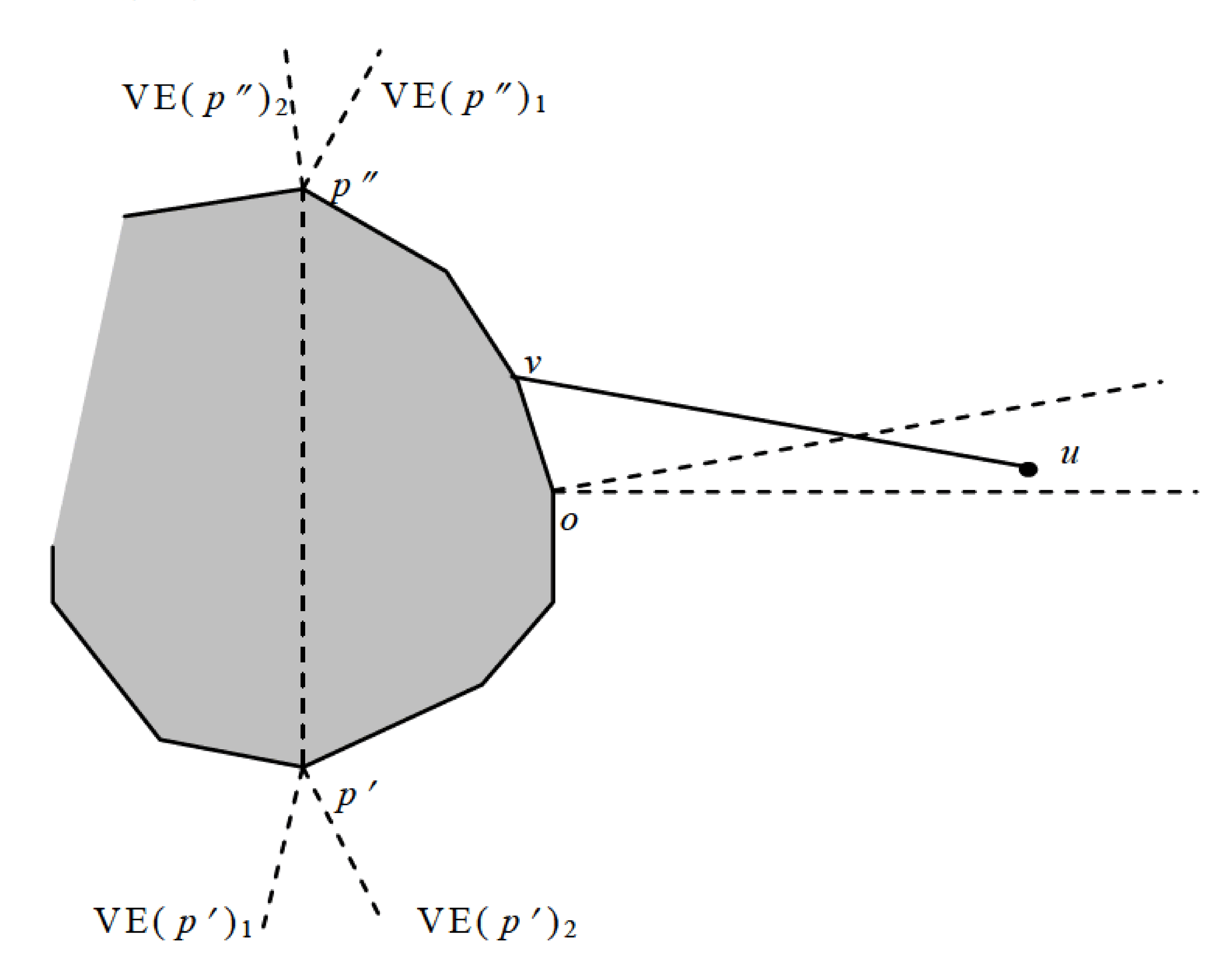


图 3.14　P 外一点 u 与多边形链 $\mathrm{C}(p', p'')$的关系

对于任意一点 $u \notin P$，我们首先取 $p'=p_0$，$p''=p_{n/2}$，然后根据引理 3.7 判断 u 是在 $C(p', p'')$中还是在 $C(p'', p')$中，然后继续在 $C(p', p'')$或 $C(p'', p')$中用二分法查找，最后可查找出 P 中离 u 最近的站点。由引理 3.7 知，对于任意一点 $u \notin P$ 和站点 $p' \in P$，$p'' \in P$，可以在 $O(1)$时间内判断出是否存在站点 $o \in C(p', p'')$，满足 $u \in VR(o)$。因此，对于任意一点 $u \notin P$，可在 $O(\log n)$时间内找到站点 $o \in P$，$u \in VR(o)$。因此，根据引理 3.6，对于任意一点 $u \in P$，可在 $O(\log n)$时间内找到站点 $v \in Q_1$，满足 $u \in VR(v)$。

引理 3.8 给定一个顶点 $o \in Q_d$ 和任意顶点 $o' \in Q$，$o' \neq o$。有 $o' \in Q_u$，或 $o' \in Q_m$，如果其同时满足下列四个条件：

（1） $(VE(o)_1 \times VE(o')_1) \cdot k < 0$；

（2） $(VE(o)_1 \times VE(o')_2) \cdot k < 0$；

（3） $VE(o)_1 \cdot VE(o')_1 > 0$；

（4） $VE(o)_1 \cdot VE(o')_2 < 0$。

证明： 假设 o' 同时满足四个条件，但 $o' \in Q_2$。由于 $o' \in Q_2$，存在 $(VE(o)_1 \times VE(o')_1) \cdot k > 0$ 且 $(VE(o)_1 \times VE(o')_2) \cdot k > 0$，或 $VE(o)_1 \cdot VE(o')_1 < 0$ 且 $VE(o)_1 \cdot VE(o')_2 < 0$。矛盾，假设不成立。

假设 o' 同时满足四个条件，但 $o' \in Q_d$。由于 $o' \in Q_d$，所以 $VE(o)_1 \cdot VE(o')_1 > 0$ 且 $VE(o)_1 \cdot VE(o')_2 > 0$。矛盾，假设不成立。

因此，原命题成立。证毕。

类似地，可得到引理 3.9。

引理 3.9 给定一个顶点 $o \in Q_u$ 和任意顶点 $o' \in Q$，$o' \neq o$。有 $o' \in Q_d$ 或 $o' \in Q_m$，如果其同时满足下列四个条件：

（1） $(VE(o)_2 \times VE(o')_1) \cdot k > 0$；

（2） $(VE(o)_2 \times VE(o')_2) \cdot k > 0$；

（3）$\mathrm{VE}(o)_2 \cdot \mathrm{VE}(o')_1 > 0$；

（4）$\mathrm{VE}(o)_2 \cdot \mathrm{VE}(o')_2 < 0$。

定理 3.9　设 p_i、q_j 分别是 P_1、Q_1 的顶点，$p_i \notin \mathrm{VR}(q_j)$ 且 $q_j \in \mathrm{VR}(p_i)$，则有：

（1）如果 p_i 在 $\mathrm{VE}(q_j)_2$ 的左侧，则 $p_i \notin P_d$，$q_j \notin Q_d$；

（2）如果 p_i 在 $\mathrm{VE}(q_j)_1$ 的右侧，则 $p_i \notin p_u$，$q_j \notin Q_u$。

证明（反证法）：

（1）p_i 在 $\mathrm{VE}(q_j)2$ 的左侧。

（a）假设 $p_i \in P_d$。由于 $q_j \in \mathrm{VR}(p_i)$，所以有 $(\mathrm{v}_\mathrm{u}^* \times l_{i1}) \cdot k < (\mathrm{v}_\mathrm{u}^* \times p_i q_j) \cdot k < (\mathrm{v}_\mathrm{u}^* \times l_{i2}) \cdot k < 0$，即 $q_j \in Q_d$。过 q_j 作 p^*q^* 的平行线 l'，则 p_i 位于 l' 的左侧。由于 $q_j \in Q_d$，所以 $\mathrm{VR}(q_j)_1$ 和 $\mathrm{VE}(q_j)_2$ 在 l' 的右侧，因而 p_i 在 $\mathrm{VE}(q_j)_2$ 的右侧，矛盾。故 $p_i \notin P_d$。

（b）假设 $q_j \in Q_d$。由（a）知 $p_i \notin P_d$，即 $p_i \in P_m$ 或 $p_i \in P_u$。由于 $q_j \in \mathrm{VR}(p_i)$，且 $q_j \in Q_d$，所以有 $(\mathrm{v}_\mathrm{u}^* \times l_{i1}) \cdot k < (\mathrm{v}_\mathrm{u}^* \times p_i q_j) \cdot k < 0$。过 q_j 作 p^*q^* 的平行线 l'，则 p_i 位于 l' 的左侧。由于 $q_j \in Q_d$，所以 $\mathrm{VE}(q_j)_1$ 和 $\mathrm{VE}(q_j)_2$ 在 l' 的右侧，因而 p_i 在 $\mathrm{VE}(q_j)_2$ 的右侧，矛盾。故 $q_j \notin Q_d$。

（2）证明方法类似（1）。证毕。

如果 p_i 是 P_1 的一个边，则根据顶点 q_j 在 p_i 边上的投影 $p(q_j, p_i)$ 与 $\mathrm{VE}(q_j)_2$ 和 $\mathrm{VE}(q_j)_1$ 的位置关系确定 p_i 与 q_j 的位置。

定理 3.10　设站点 $o_1 \in P$，站点 $o_2 \in Q$，有 $o_2 \in \mathrm{VR}(o_1)$。

（1）如果 $o_1 \in P_d$，则 $o_2 \in Q_d$；

（2）如果 $o_1 \in P_u$，则 $o_2 \in Q_u$；

（3）如果 $o_2 \in Q_d$，则 $o_1 \in P_d$ 或 P_m；

（4） 如果 $o_2 \in Q_u$，则 $o_1 \in P_u$ 或 P_m。

证明：（1）假设 $o_1 \in P_d$，$o_2 \notin Q_d$。令 o_1^*，o_2^*表示 o_1，o_2 的最短距离点对，$o^*=o_2^*-o_1^*$为向量，则$(v^* \times o^*) \bullet k>0$。根据性质 3.3，$o_2$ 在 $VE(o_1)_2$ 的左侧，$o_2 \notin VR(o_1)$。矛盾。所以，如果 $o_1 \in P_d$，则 $o_2 \in Q_d$。

类似的方法可证明（2）、（3）、（4）也成立。证毕。

算法 3.3　计算 P 和 Q 的初始搜索范围 P'和 Q'

（1）取 P 的第一个顶点 p_1，根据引理 3.7，用二分法在 Q 中计算站点 o_1，其中 $p_1 \in VR(o_1)$；

（1.1）如果 o_1 是边，取 q_{i2} 为 o_1 的一个端点；如果 o_1 是顶点，$q_{i2}=o_1$；

（1.2）根据引理 3.6 知，$q_{i2} \in Q_1$。

（2）类似地，用二分法在 P 中计算站点 o_2，其中 $q_{i2} \in VR(o_2)$；

如果 o_2 是顶点且 $o_2 \in VR(q_{i2})$，或 o_2 是边且 $p(q_{i2}, o_2) \in VR(q_{i2})$，则 o_2、q_{i2} 为最短距离站点对，整个算法结束，否则转（3）。

（3）根据定理 3.9，确定 q_{i2} 的位置：

（3.1）如果 o_2 在 $VE(q_{i2})_2$ 的左侧，则 $o_2 \notin P_d$，$q_{i2} \notin Q_d$；

（3.2）如果 o_2 在 $VE(q_{i2})_1$ 的右侧，则 $o_2 \notin P_u$，$q_{i2} \notin Q_u$。

（4）根据引理 3.8 和引理 3.9，用二分法在 Q 中计算 q_{j2}：

（4.1）如果 $q_{i2} \notin Q_d$：

（4.1.1）如果 $q_{(i2+2^k)} \in Q_d$ 或 Q_m，则 $q_{j2}=q_{(i2+2^k)}$（k 为整数，初值为 0）；

（4.1.2）如果 $q_{(i2+2^k)} \in Q_u$，则继续判断 $q_{(i2+2^{k+1})}$的位置；

（4.1.3）否则，在 $q_{(i2+2^k)}$与 $q_{(i2+2^{k+1})}$之间，用类似的方法（索引每次增加 2、4、8…）进行查找，可得到 $q_{j2} \in Q_d$ 或 Q_m。

（4.2）如果 $q_{i2}\notin Q_u$，可沿顺时针方向用上面类似的方法确定 $q_{j2}\in Q_u$ 或 Q_m。

（5）用二分法在 P 中计算 o_4，其中 $q_{j2}\in \mathrm{VR}(o_4)$；

如果 o_4 是顶点且 $o_4\in \mathrm{VR}(q_{j2})$，或 o_4 是边且 $p(q_{j2}, o_4)\in \mathrm{VR}(q_{j2})$，则 o_4、q_{j2} 为最短距离特征对，整个算法结束。

（6）根据定理 3.10 和 q_{i2}、q_{j2} 的位置，确定 p_{i1}、p_{j1} 分别取 o_2、o_4 的哪个端点，并确定 p_{i1}，p_{i2} 的位置。

p_{i1}、p_{j1} 间的站点组成 P'（逆时针），q_{i2}、q_{j2} 间的站点组成 Q'（逆时针）。根据引理 3.8、引理 3.9 以及定理 3.10，P'和 Q'即为包含有最短距离站点对的初始搜索范围。这里，P'和 Q'分别只是 P_1、Q_1 的一部分。

由前面的分析知，算法 3.3 计算 q_{i2} 的时间复杂度为 $O(\log m)$，计算 p_{i1} 的时间复杂度为 $O(\log n)$；根据引理 3.9，计算 q_{j2} 的时间复杂度为 $O(\log m)$，计算 p_{j1} 的时间复杂度为 $O(\log n)$。因此，算法 3.3 中确定初始搜索范围的时间复杂度为 $O(\log n+\log m)$。

3. 计算候选站点的位置

下面讨论关键问题 2 的解决方法。

设两个顶点 $p_i\in P_1$，$q_j\in Q_1$。p_i、q_j 关于 $\mathrm{VR}(p_i)$、$\mathrm{VR}(q_j)$的位置关系可以分为以下几种情况。

情况 1： q_j 在 $\mathrm{VE}(p_i)_1$ 的右侧，又分为以下三种子情况：

情况 1.1： p_i 在 $\mathrm{VE}(q_j)_2$ 的左侧；

情况 1.2： $p_i\in \mathrm{VR}(q_j)$，即 p_i 在 $\mathrm{VE}(q_j)_1$ 的左侧，且在 $\mathrm{VE}(q_j)_2$ 的右侧；

情况 1.3： p_i 在 $\mathrm{VE}(q_j)_1$ 的右侧。

情况 2： $q_j\in \mathrm{VR}(p_i)$，即 q_j 在 $\mathrm{VE}(p_i)_1$ 的左侧且在 $\mathrm{VE}(p_i)_2$ 的右侧，又分为以下三种子情况：

情况 2.1：p_i 在 $\mathrm{VE}(q_j)_2$ 的左侧；

情况 2.2：$p_i \in \mathrm{VR}(q_j)$，即 p_i 在 $\mathrm{VE}(q_j)_1$ 的左侧，且在 $\mathrm{VE}(q_j)_2$ 的右侧；

情况 2.3：p_i 在 $\mathrm{VE}(q_j)_2$ 的右侧。

情况 3：q_j 在 $\mathrm{VE}(p_i)_2$ 的左侧，又分为以下三种子情况：

情况 3.1：p_i 在 $\mathrm{VE}(q_j)_1$ 的右侧；

情况 3.2：$p_i \in \mathrm{VR}(q_j)$，即 p_i 在 $\mathrm{VE}(q_j)_1$ 的左侧，且在 $\mathrm{VE}(q_i)_2$ 的右侧；

情况 3.3：p_i 在 $\mathrm{VE}(q_j)_2$ 的左侧。

如图 3.15 所示，q_7 在 $\mathrm{VE}(p_1)_1$ 的右侧，q_3、q_4、q_5、q_6、q_7 在 $\mathrm{VE}(p_2)_1$ 的右侧，q_1、q_2、q_3、q_4、q_5 在 $\mathrm{VE}(p_1)_2$ 的左侧，q_1 在 $\mathrm{VE}(p_2)_2$ 的左侧，q_6 在 $\mathrm{VR}(p_1)$ 中，q_2 在 $\mathrm{VR}(p_2)$中，p_2 在 $\mathrm{VE}(q_1)_2$、$\mathrm{VE}(q_2)_2$、$\mathrm{VE}(q_3)_2$ 的左侧，p_2 在 $\mathrm{VR}(q_4)$ 中，p_1、p_2 在 $\mathrm{VE}(q_5)_1$、$\mathrm{VE}(q_6)_1$、$\mathrm{VE}(q_7)_1$ 的右侧，p_1 在 $\mathrm{VE}(q_1)_2$、$\mathrm{VE}(q_2)_2$、$\mathrm{VE}(q_3)_2$、$\mathrm{VE}(q_4)_2$ 的左侧，$\langle p^*, q^* \rangle$为 P 和 Q 的一个最短距离实现点对。

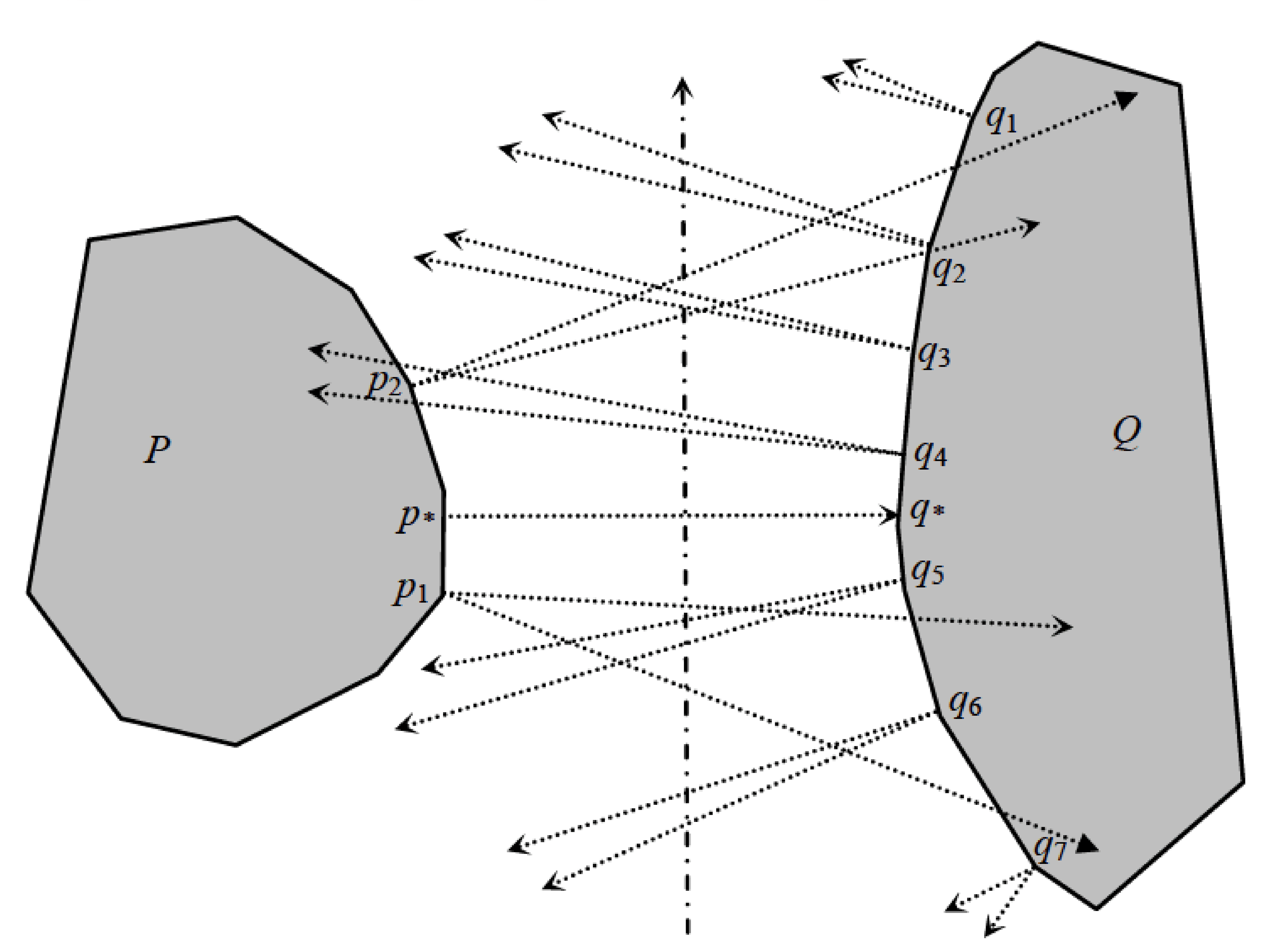


图 3.15　候选站点间的位置关系

定理 3.11　设 q_j 在 $\mathrm{VE}(p_i)_1$ 的右侧，p_i 在 $\mathrm{VE}(q_j)_1$ 的右侧，则有：

（1）如果$(\mathrm{VE}(p_i)_1 \times \mathrm{VE}(q_j)_1) \bullet k \geqslant 0$，则 $p_i \in P_u$；

（2）如果$(VE(p_i)_1 \times VE(q_j)_1) \bullet k<0$，则 $q_j \in Q_d$。

证明：（1）假设$(VE(p_i)_1 \times VE(q_j)_1) \bullet k \geqslant 0$，但 $p_i \in P_d$。由于 q_j 在 $VE(p_i)_1$ 的右侧，p_i 在 $VE(q_j)_1$ 的右侧，所以 $q_j \in Q_d$。根据性质 3.4，有$(VE(p_i)_1 \times VE(q_j)_1) \bullet k<0$。矛盾。所以原命题成立。类似地，可证明（2）。证毕。

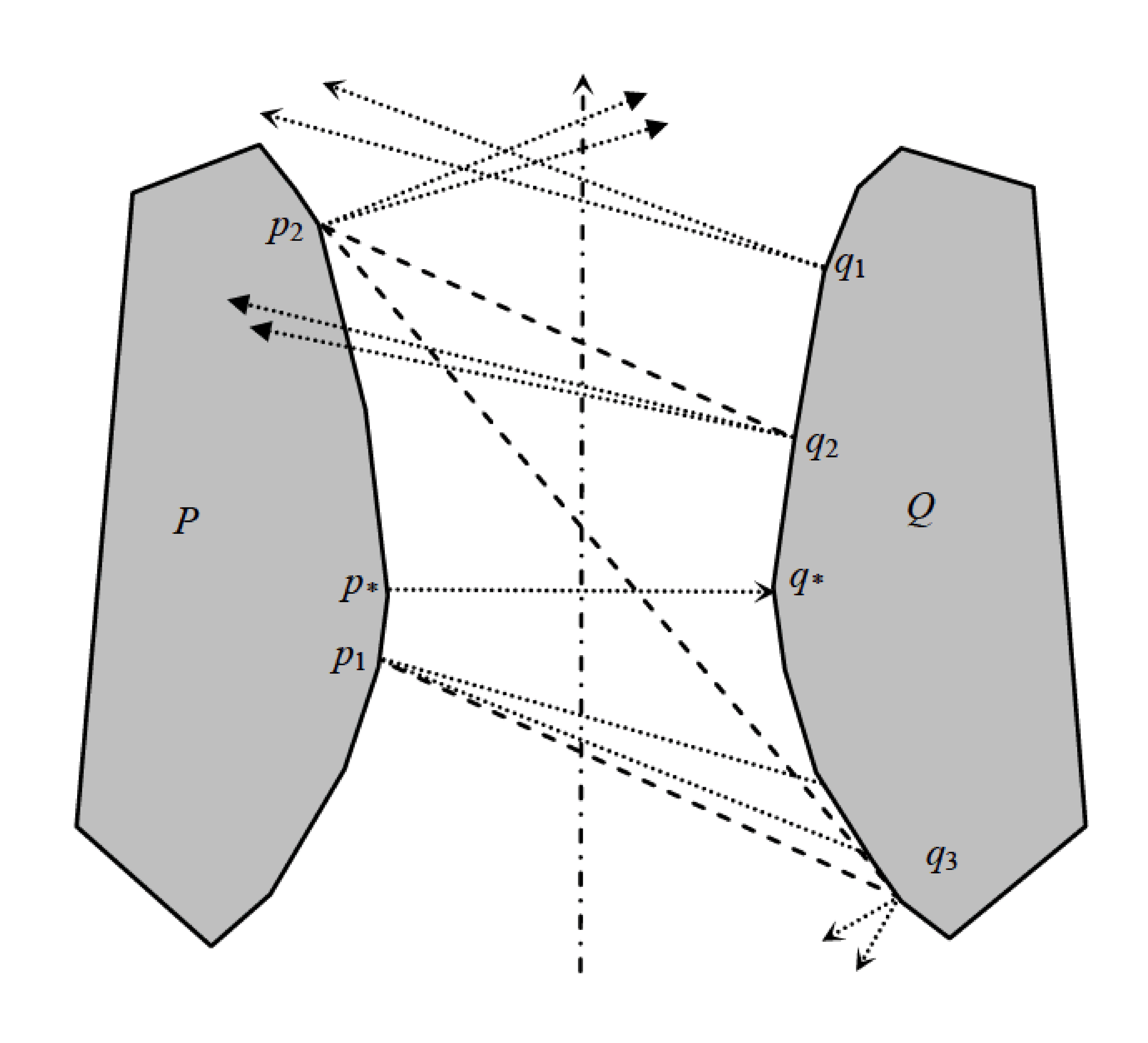


图 3.16　情况 1.3 中候选站点间的位置关系

对于情况 1.3（q_j 在 $VE(p_i)_1$ 的右侧，p_i 在 $VE(q_j)_1$ 的右侧）：当 $p_i \in P_u$ 时，$q_j \in Q_u$，Q_m 或 Q_d（如图 3.16 中的 p_2 与 q_2，p_2 与 q_3，其中 $p_2 \in P_u$，$q_2 \in Q_u$，$q_3 \in Q_d$）。当 $p_i \in P_m$ 或 P_d 时，$q_j \in Q_d$（如图 3.16 中的 p_1 与 q_3，其中 $p_1 \in P_d$，$q_3 \in Q_d$）。如果$(VE(p_2)_1 \times VE(q_2)_1) \bullet k \geqslant 0$，则可排除 $p_i \in P_m$ 且 $q_j \in Q_d$，$p_i \in P_d$ 且 $q_j \in Q_d$ 的情况（如图 3.16 中的 p_1 与 q_3），因此必有 $p_i \in P_u$（如图 3.16 中的 p_2 与 q_2，p_2 与 q_3）。如果$(VE(p_i)_1 \times VE(q_j)_1) \bullet k <0$，则可排除 $q_j \in Q_u$ 和 $q_j \in Q_m$ 的情况，因此必有 $q_j \in Q_d$（如图 3.16 中的 p_1 与 q_3）。

定理 3.12　设 q_j 在 $VE(p_i)_2$ 的左侧，p_i 在 $VE(q_j)_2$ 的左侧，则有：

（1）如果$(VE(q_j)_2 \times VE(p_i)_2) \bullet k \leqslant 0$，则 $q_j \in Q_u$；

（2）如果$(VE(q_j)2 \times VE(p_i)_2) \bullet k>0$，则 $p_i \in P_d$。

证明方法类似定理 3.11。

对于情况 3.3（q_j 在 $VE(p_i)_2$ 的左侧，p_i 在 $VE(q_j)_2$ 的左侧）：当 $p_i \in P_u$ 或 P_m 时，$q_j \in Q_u$（如图 3.17 中的 p_2 与 q_1，其中 $p_2 \in P_u$，$q_1 \in Q_u$）。当 $p_i \in P_d$ 时，$q_j \in Q_u$，Q_m 或 Q_d（如图 3.17 中的 p_1 与 q_1，p_1 与 q_2，其中 $p_1 \in P_d$，$q_1 \in Q_u$，$q_2 \in Q_d$）。如果$(VE(q_j)_2 \times VE(p_i)_2) \bullet k \leqslant 0$，可排除 $q_j \in Q_u$ 和 $q_j \in Q_m$，必有 $q_j \in Q_u$。类似地，如果$(VE(q_j)_2 \times VE(p_i)_2) \bullet k > 0$，可排除 $p_i \in P_u$ 和 $p_i \in P_m$，则 $p_i \in P_d$。

针对上面的每一种情况，我们给出判断 p_i，q_j 关于有向线段 p^*q^*的位置的算法。

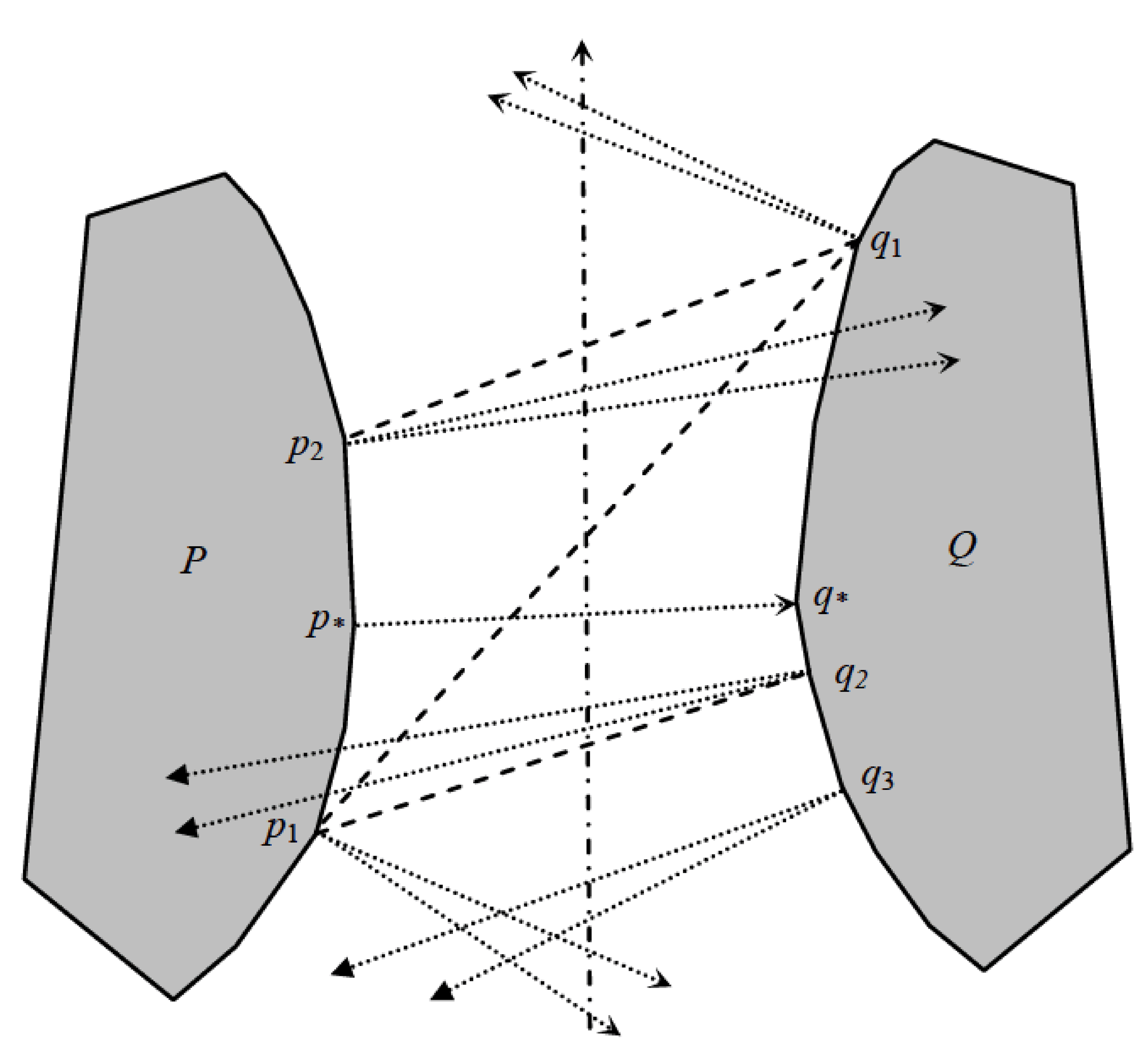


图 3.17　情况 3.3 中候选站点间的位置关系

算法 3.4　判断 p_i、q_j 关于有向线段 p^*q^*的位置

(1) 如果满足情况 1：

（1.1）如果满足情况 1.1，则 $p_i \in P_u$，$q_j \in Q_u$；

（1.2）如果满足情况 1.2，则 $p_i \in P_u$，$q_j \in Q_u$；

（1.3）如果满足情况 1.3，根据定理 3.11，有：

（a）如果$(\mathrm{VE}(p_i)_1 \times \mathrm{VE}(q_j)_1) \bullet k \geqslant 0$，则 $p_i \in P_u$；

（b）如果$(\mathrm{VE}(p_i)_1 \times \mathrm{VE}(q_j)_1) \bullet k < 0$，则 $q_j \in Q_d$。

（2）如果满足情况 2：

（2.1）如果满足情况 2.1，则 $p_i \in P_u$ 或 $p_i \in P_m$，$q_j \in Q_u$；

（2.2）如果满足情况 2.2，则 $p_i \in P_m$，$q_j \in Q_m$；

（2.3）如果满足情况 2.3，则 $p_i \in P_d$ 或 $p_i \in P_m$，$q_j \in Q_d$。

（3）如果满足情况 3：

（3.1）如果满足情况 3.1，则 $p_i \in P_d$，$q_j \in Q_d$；

（3.2）如果满足情况 3.2，则 $p_i \in P_d$，$q_j \in Q_d$；

（3.3）如果满足情况 3.3，根据定理 3.12，有：

（a）如果$(\mathrm{VE}(q_j)_2 \times \mathrm{VE}(p_i)_2) \bullet k > 0$，则 $p_i \in P_d$。

（b）如果$(\mathrm{VE}(q_j)_2 \times \mathrm{VE}(p_i)_2) \bullet k \leqslant 0$，则 $q_j \in Q_u$。

根据算法 3.4，可以确定 p_i 或 q_j 关于 p^*q^* 的位置，然后可以确定新的搜索范围。算法 3.4 判断 $p_i \in P_1$ 和 $q_j \in Q_1$ 位置的时间复杂度为 $O(1)$。

4. 计算最短距离站点对

为便于描述，设初始搜索范围 P'的起点 $p' \in P_d$，终点 $p'' \in P_u$；Q'的终点 $q' \in Q_d$，起点 $q'' \in Q_u$。

算法 3.5　查找最短距离站点对

（1）如果 p'与 p''为相邻的两个顶点，则可用二分法在 Q'中计算 q。

（1.1）如果 q 是顶点，满足 $o^* \in \mathrm{VR}(q)$且 $q \in \mathrm{VR}(o)$，其中 o 为站点 p'、p''或 $p'p''$，o^*为 q 到 p'、p''或 $p'p''$的投影；则算法结束。

（1.2）如果 q 是边，满足 $p'\in\mathrm{VR}(q)$且 $q^*\in\mathrm{VR}(p')$，或 $p''\in\mathrm{VR}(q)$且 $q^*\in\mathrm{VR}(p'')$，其中 q^*为 p'或 p''到 q 的投影；则算法结束。

（2）如果 q'与 q''为相邻的两个顶点，则可用二分法在 P'中计算 p。

（2.1）如果 p 是顶点，满足 $o^*\in\mathrm{VR}(p)$且 $p\in\mathrm{VR}(o)$，其中 o 为站点 q'、q''或 $q'q''$，o^*为 p 到 q'、q''或 $q'q''$的投影；则算法结束。

（2.2）如果 p 是边，满足 $q'\in\mathrm{VR}(p)$且 $p^*\in\mathrm{VR}(q')$，或 $q''\in\mathrm{VR}(p)$且 $p^*\in\mathrm{VR}(q'')$，其中 p^*为 q'或 q''到 p 的投影；则算法结束。

（3）分别取 P'、Q'中间的顶点站点 p_i、q_j，它们将 P'，Q'分成子多边形链 P_1'' 和 P_2''、Q_1'' 和 Q_2''。

（4）用算法 3.4 判断 p_i 和 q_j 位置。

（5）确定新的搜索范围。

（5.1）如果 $p_i\in P_m$ 且 $q_j\in Q_m$，则 $<p_i, q_j>$为最短距离站点对；算法结束；

（5.2）如果 $p_i\in P_m$ 且 $q_j\notin Q_m$，则找到最短距离站点对中的一个 $p=p_i$，然后在 Q'中用二分法计算 q，其中 $p\in\mathrm{VR}(q)$且 $q^*\in\mathrm{VR}(p)$，算法结束；

（5.3）如果 $q_j\in Q_m$ 且 $p_i\notin P_m$，则找到最短距离站点对中的一个 $q=q_j$，然后在 P'中用二分法计算 p，其中 $p^*\in\mathrm{VR}(q)$且 $q\in\mathrm{VR}(p)$，算法结束；

（5.4）如果 $p_i\in P_u$，则 $P'=P_1''$，转（1）；

（5.5）如果 $p_i\in P_d$，则 $P'=P_2''$，转（1）；

（5.6）如果 $q_j\in Q_u$，则 $Q'=Q_1''$，转（1）；

（5.7）如果 $q_j\in Q_d$，则 $Q'=Q_2''$，转（1）。

算法 3.5 的时间复杂度为 $O(\log n+\log m)$。

算法 3.6　计算两个分离凸多边形间的距离

（1）用算法 3.3 计算 P 和 Q 的初始搜索范围 P'和 Q'；

（2）用算法 3.5 在 P'和 Q'中查找一个最短距离站点对$<o_p, o_q>$；

（3）根据$<o_p, o_q>$的类型计算 P 和 Q 的距离：$d(P, Q)=d(o_p, o_q)$。

由推论 3.4 可知，对于凸多边形的每个站点的外部 Voronoi 区域只有 2 条 Voronoi 边，且每条 Voronoi 边都和多边形的边垂直。因此，计算一个站点的 Voronoi 边的时间为 $O(1)$，这种计算只在计算最短距离站点对的过程中进行，不需要预先计算与存储。由上面对算法 3.3 和算法 3.5 的分析知，计算初始搜索范围及在其中查找一个最短距离站点对的时间复杂度都为 $O(\log n+\log m)$。另外，根据最短距离站点对计算多边形的距离的时间为 $O(1)$。因此，整个算法的时间复杂度为 $O(\log n+\log m)$。

3.3.3　简单多边形中的最短路径计算

本节介绍一种基于 Voronoi 图的计算简单多边形中任意两点 s 和 t 之间的最短路径 SP(s，t)的算法[YangCL2007]。

1. 概念和性质

在介绍具体算法之前，先介绍几个基本概念。

简单多边形 P 中两点 s 和 t 之间的 Voronoi 骨架路径 VSP(s，t)是一条由系列 Voronoi 边首尾依次连接而成的路径。如图 3.18 所示，其由三个部分组成，分别是中间骨架部分 $s_1v_1r_1\ldots r_kv_2t_2$，起始部分 ss_1 以及结束部分 t_2t。其中，由 s 点作相应多边形边（s 所在 Voronoi 区域所对应的边）的垂线（如果 s 在内尖点 p 的 Voronoi 区域内，则直接连接 sp），交 Voronoi 边于 s_1，从而得到路径的起始部分 ss_1；由 t 点作相应多边形边的垂线（如果 t 在内尖点 q 的 Voronoi 区域内，则直接连接 qt），交 Voronoi 边于 t_2，得到路径的结束部分 t_2t。又如图 3.19 中粗虚线所示，VSP(v，p_9)={v，a，…，o，p_9}。

若 Voronoi 边 e 为两 Voronoi 区域 VR(o_1)和 VR(o_2)的边界所共有，则称

VR(o_1)和 VR(o_2)为 e 的关联 Voronoi 区域，o_1 和 o_2 为 e 的关联边。假设 a 和 b 分别为 e 的起点和终点，若 o_1 在有向边 ab 的左侧(右侧)，则称 o_1 为 ab 的左侧(右侧)关联边。在图 3.19 中，边 p_1p_2 为有向边 de 的左侧关联边，凹顶点(特殊边)p_{18} 为有向边 de 的右侧关联边。需要注意的是，p_1p_2 为有向边 ed 的右侧关联边，p_{18} 为有向边 ed 的左侧关联边。

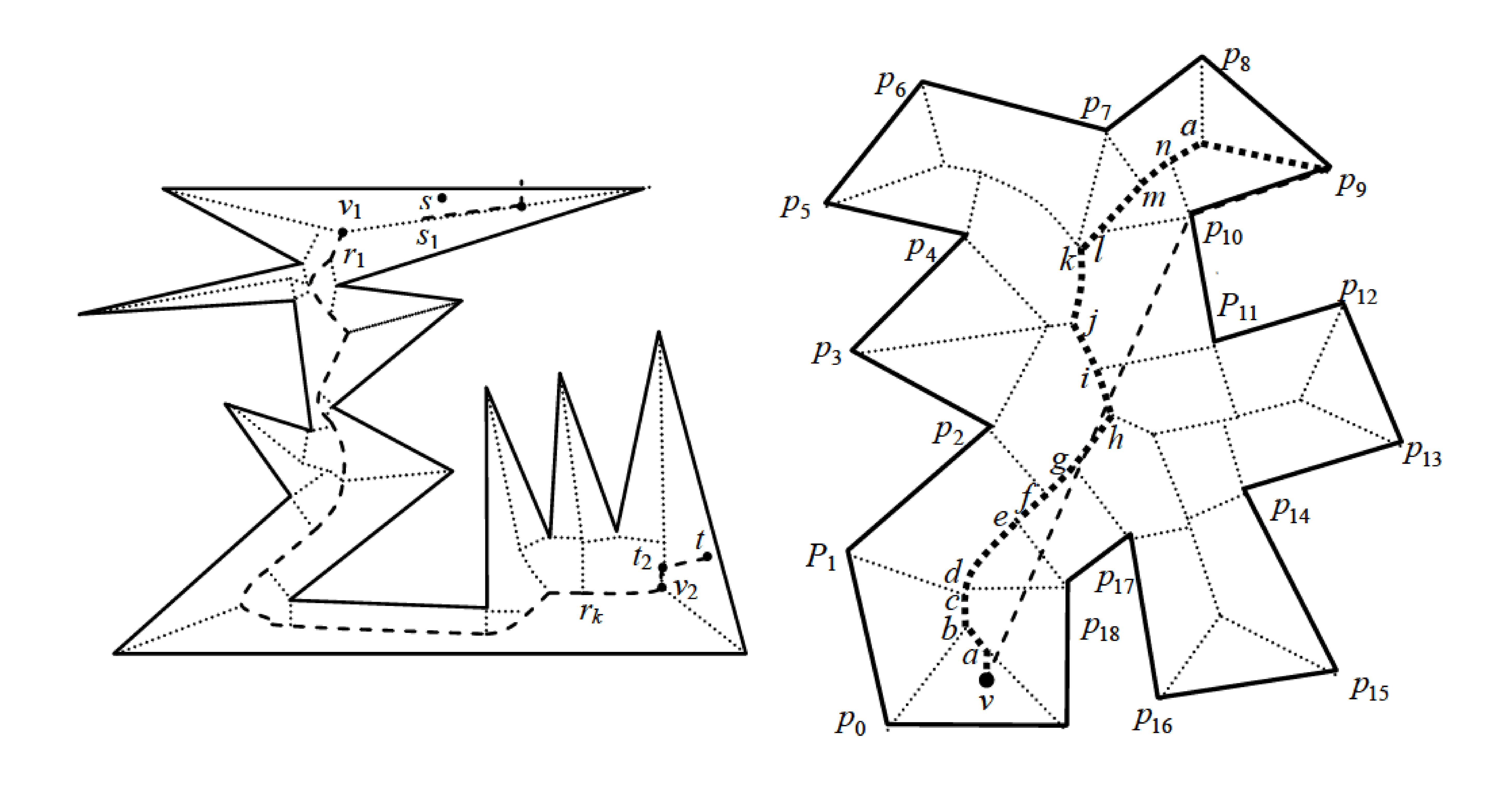


图 3.18　Voronoi 骨架路径　　　　图 3.19　简单多边形中的 Voronoi 骨架路径(粗虚线)

如果一个内尖点 r 的 Voronoi 区域与一条 Voronoi 骨架路径有公共部分（Voronoi 边)，则称 r 为该路径的关联内尖点。当我们沿着 Voronoi 骨架路径走过时，发现所有的关联内尖点会依次路过，那些位于路径左侧的关联内尖点称为左关联内尖点，相应地也可以定义右关联内尖点。

设 s、t 是多边形 P 内的两点，p_0, p_1, …, p_n(p_0=s, p_n=t)是最短路径 SP(s, t)上的所有点。可以看出最短路径是由左关联内尖点和右关联内尖点组成的。我们得到以下引理。

引理 3.10　SP(s, t)上的任意顶点 p_i ($0<i<n$)一定是路径 VSP(s, t)上的关联内尖点。

证明： 图 3.20 中 *sABCDt* 是最短路径，存在且只存在一条 Voronoi 骨架路径连接 *s* 与 *t*。我们证明最短路径上的任意一点 *B* 都是该 Voronoi 骨架路径的一个关联内尖点。

假定 *BF* 是∠*CBA* 的平分线，*F* 是 *BF* 与 Voronoi 骨架路径 *EFGH* 的交点。从点 *F* 看，*KLBCMN* 的左侧可见部分在点画线 *ABCD* 之后或者上面。因为 *ABCD* 是一个凸包链，而点 *F* 到 *KLBCMN* 的最近点是 *B*，所以 *B* 点一定是一个关联内尖点。证毕。

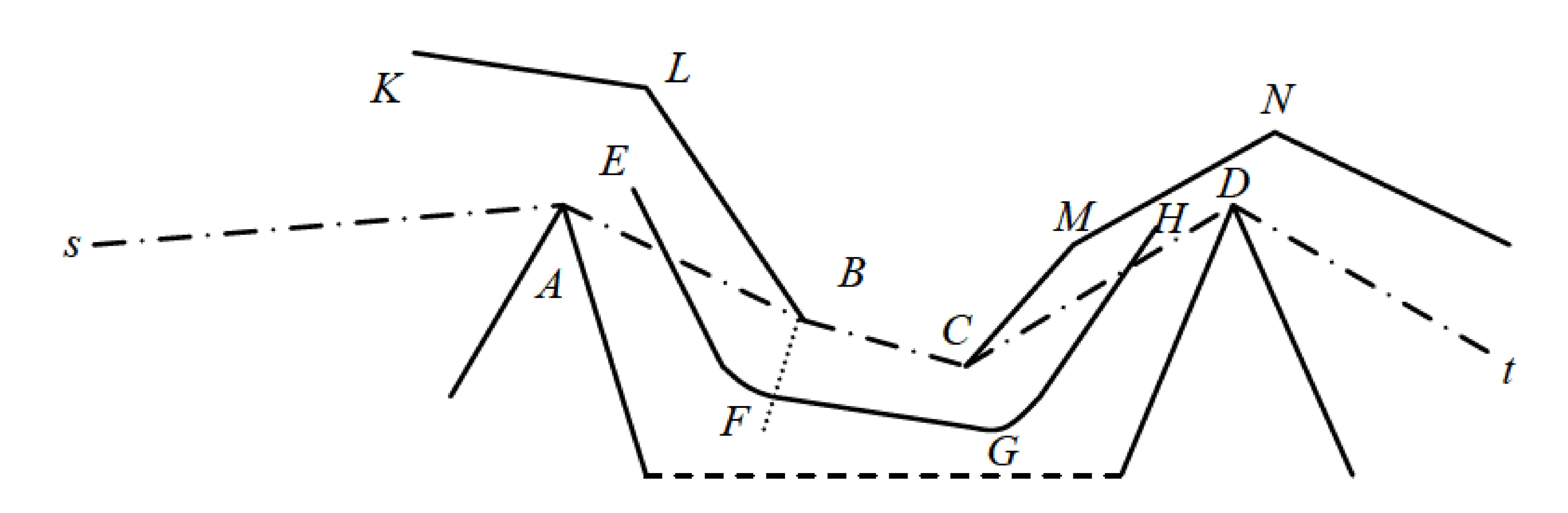


图 3.20　*s*、*t* 之间的最短路径（点画线）

设 q_1，q_2，…，q_m 是 VSP(*s*，*t*)的关联内尖点，是沿 VSP(*s*，*t*)访问依次得到的，而最短路径 SP(*s*，*t*)是由其中的点 q_{k1}，q_{k2}，…，q_{kn} 依次组成。由引理 3.10 知，q_{k1}，q_{k2}，…，q_{kn} 是 q_1，q_2，…，q_m 的子集。

引理 3.11　点 q_{k1}，q_{k2}，…，q_{kn} 在有序序列 q_1，q_2，…，q_m 中的顺序与在 SP(*s*，*t*)中的顺序一致。

证明： 如果最短路径上的点都是左关联内尖点或者都是右关联内尖点，则证明显然成立。所以只需要证明最短路径上的点同时来自左、右内尖点时的情况，如图 3.21 中的点 *A*、*B*。

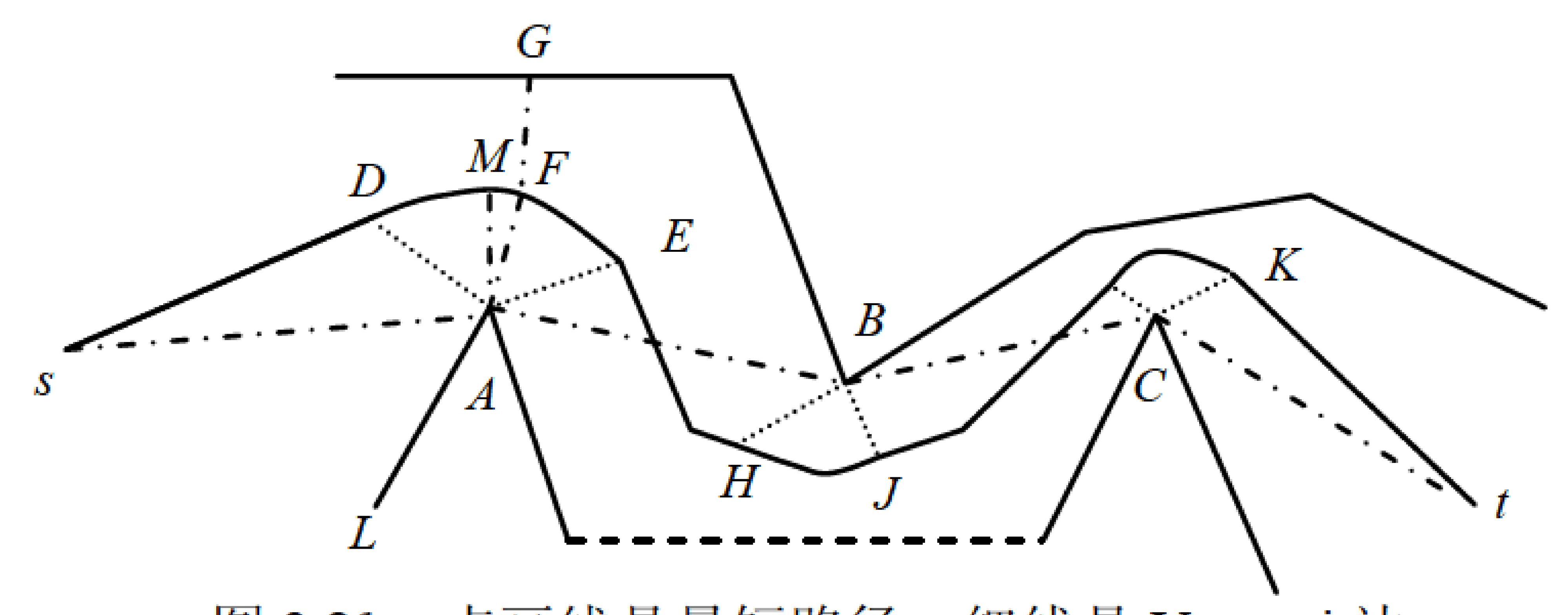


图 3.21　点画线是最短路径，细线是 Voronoi 边

在图 3.21 中，*sABCt* 是最短路径，细线 *sDFEHJKt* 是 Voronoi 骨架路径。点 *F* 是骨架上的一点，且 $AF \perp AB$。*G* 与 *A* 是距离 *F* 最近的两个点。显然，Voronoi 骨架路径 *sDFEHJKt* 仅能通过 *sDF* 一次。从点 *B* 看骨架 *sDF* 上的点一定经过 *AFG*。点 *B* 的 Voronoi 区域的边界不可能存在于 *sDF* 上，因为 *sDF* 上的点与 *B* 的距离均大于与 *A* 的距离。*M* 是 Voronoi 骨架路径上的点，且 $AM \perp As$，因为 *s* 从 *B* 点不可见，所以 *M* 在 *AF* 的左侧。区域 *AMF* 一定是点 *A* 的 Voronoi 区域的一部分。因此在 *sDF* 骨架上一定存在 *A* 的 Voronoi 区域的边界，而 *B* 的 Voronoi 区域的边界只能存在于 *FEHJKt* 上。这就证明了 Voronoi 骨架上的内尖点顺序与最短路径上的内尖点顺序一致。证毕。

通过引理 3.10 与引理 3.11 可以得到定理 3.13。

定理 3.13 SP(*s*，*t*)上的顶点可以沿着 VSP(*s*，*t*)顺序寻找，且一定是 VSP(*s*，*t*)的关联内尖点集合的子集。

由此可以判断，如果 VSP(*s*，*t*)中不包含内尖点，则 *s* 与 *t* 之间是可见的，SP(*s*，*t*)就是线段 *st*。否则，按照定理 3.13 设计算法来计算 SP(*s*，*t*)。

2. 算法思想与步骤

由于在简单多边形中 *s*、*t* 之间只存在一条 Voronoi 骨架路径 VSP(*s*，*t*)，SP(*s*，*t*)上的点能够通过顺序遍历 VSP(*s*，*t*)关联的多边形顶点求得。计算 SP(*s*，*t*)的算法有以下三个步骤。

（1）首先确定 *s*、*t* 所在的 Voronoi 区域；

（2）然后确定 VSP(*s*，*t*)；

（3）最后顺序遍历 VSP(*s*，*t*)，确定 SP(*s*，*t*)。

算法的第（1）步可以通过计算 *s* 或 *t* 到多边形的每个站点的距离，然后找出最短距离，即可确定 *s* 或 *t* 所在的 Voronoi 区域。

算法的第（2）步可以先通过 *s* 作相应多边形边的垂线（如果 *s* 在内尖点 *p* 的 Voronoi 区域内，则直接连接 *sp*），得到路径的起始部分 ss_1；同样计算出路径的结束部分 t_2t。然后通过遍历多边形的 Voronoi 图（是一棵树）得到

s_1 和 t_2 之间的 Voronoi 骨架路径，将这三部分连接起来即可得到 VSP(s，t)。

这里重点介绍如何实现第（3）步，即如何沿 VSP(s，t)计算 SP(s，t)。其计算方法是：首先，算法从 s 往 t 遍历 VSP(s，t)时，依次找出关联的左右内尖点。然后，计算已找到的左、右内尖点的凸包链 p_0，q_{l1}，q_{l2}，…及 p_0，q_{r1}，q_{r2}，…（见图 3.22），同时维护从当前点 p_0 出发的左、右切线（例如图 3.22 中的 $p_0 q_{l2}$ 和 $p_0 q_{r1}$）。接下来，检查右凸包链的切线 $p_0 q_{r1}$ 是否与左侧的关联边相交，左凸包链的切线 $p_0 q_{l2}$ 是否与右侧的关联边相交，沿左、右内尖点的交替顺序检查。如果切线 $p_0 q_{l2}$ 与边 $v_{r4} q_{r4}$ 相交，则 q_{l2} 一定是 SP(s，t)上的一点。将当前点移动到 q_{l2}，继续检查工作。由于左侧切线与右关联边 $v_{r4} q_{r4}$ 也相交，则点 q_{l3} 也一定是 SP(s，t)上的点。将当前点移动到 q_{l3}，继续如上的计算。

我们注意到，在检查过程中所用的多边形的边一定是 VSP(s，t)的关联边。为了减少运算，可以事先将不关联的边用线段截去（例如图 3.23 中的 $v_{l2} v_{l3}$）。将所有的关联边以及用于截去非关联边的线段叫做检查边。

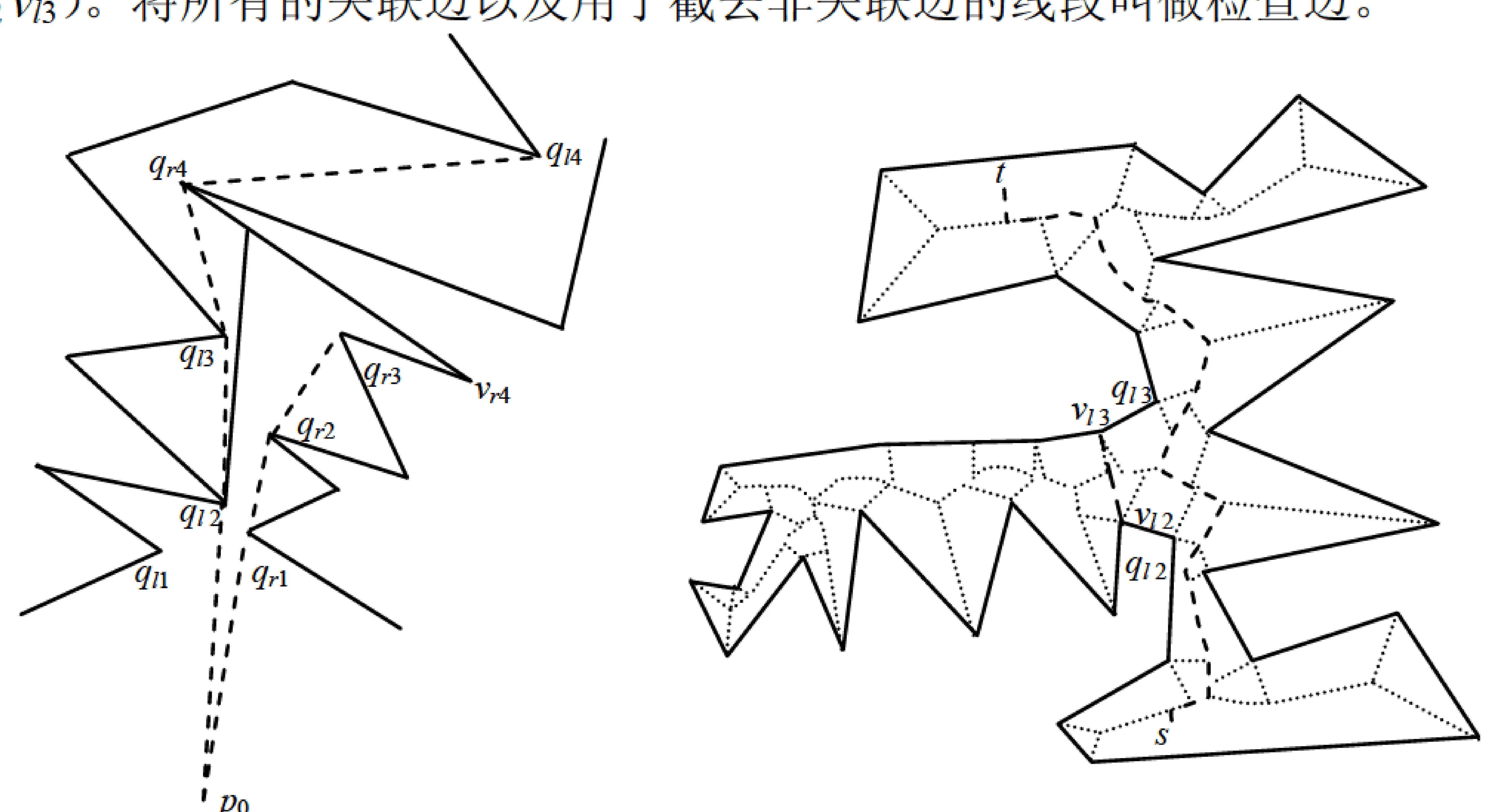
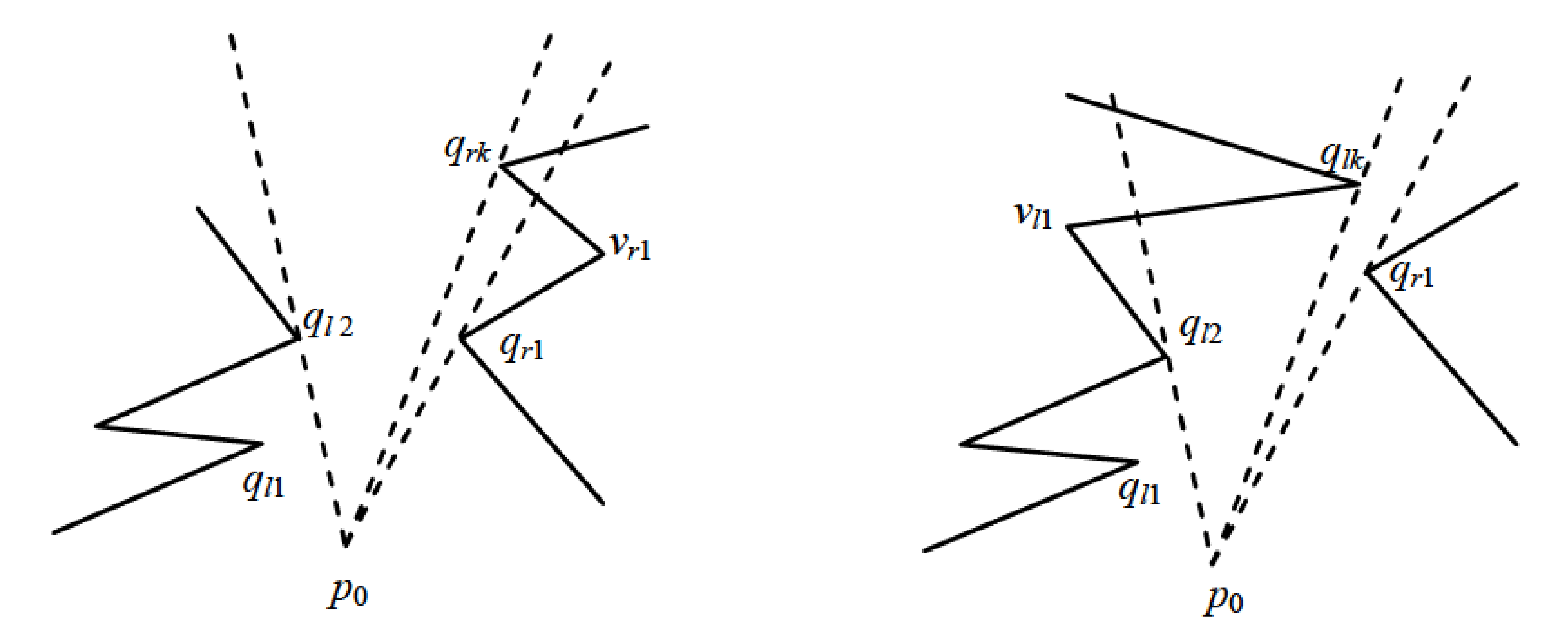
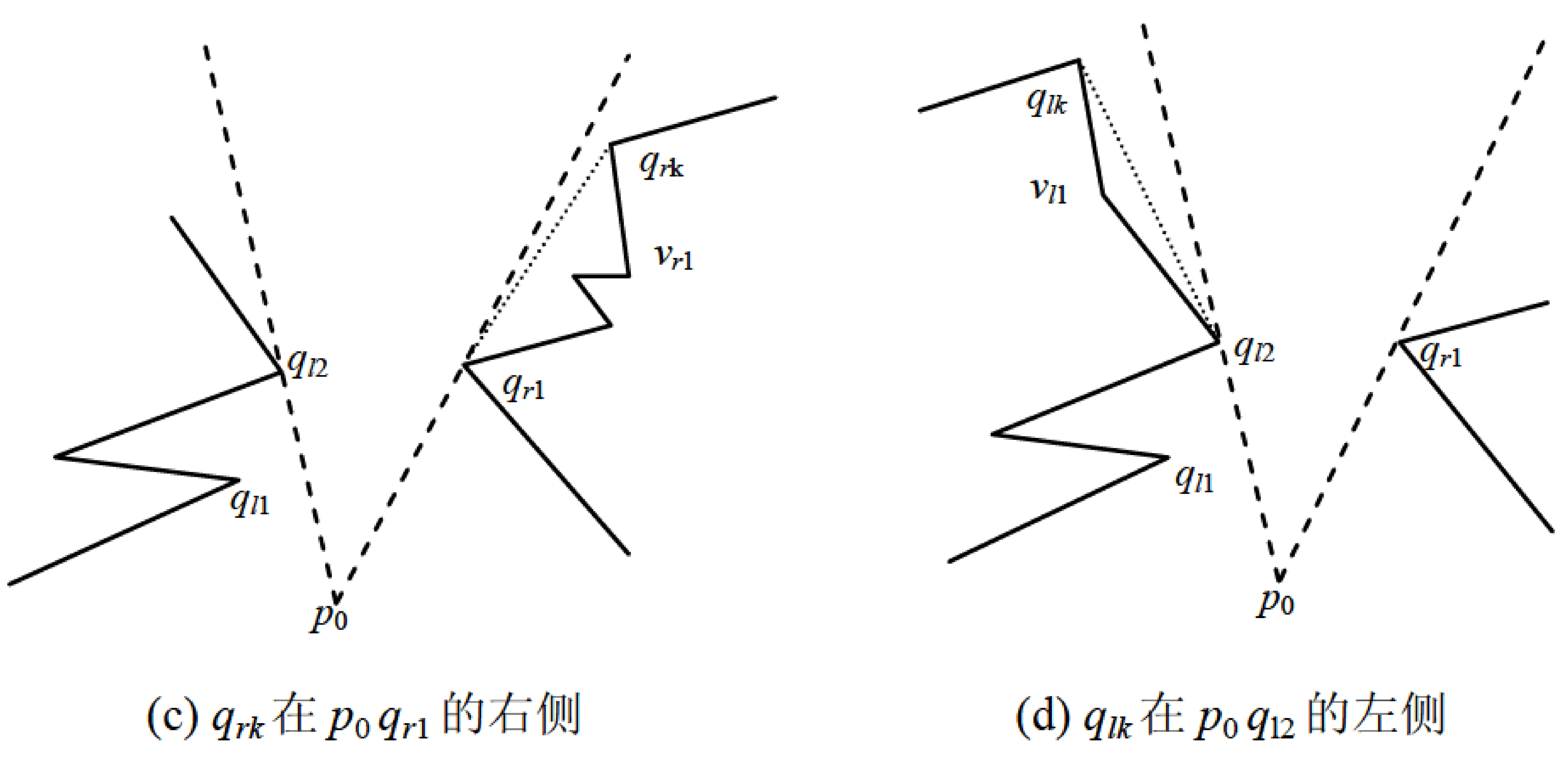


图 3.22　左切线与右侧多边形边相交　　图 3.23　去掉与 Voronoi 骨架路径不关联的多边形的边

在上述计算过程中，当出现如图 3.24（$v_{r1} q_{rk}$、$v_{l1} q_{lk}$ 是当前的检查边，$p_0 q_{l2}$、$p_0 q_{r1}$ 是当前切线）所示的情况时，凸包链以及左、右切线需要重新计算。



(a) q_{rk} 在 $p_0 q_{r1}$ 的左侧、$p_0 q_{l2}$ 的右侧　(b) q_{lk} 在 $p_0 q_{r1}$ 的左侧，$p_0 q_{l2}$ 的右侧



(c) q_{rk} 在 $p_0 q_{r1}$ 的右侧　(d) q_{lk} 在 $p_0 q_{l2}$ 的左侧

图 3.24　凸包链以及左、右切线需要重新计算

当出现图 3.25 中的情况时，q_{rk} 的两个相邻的顶点 v_{r1} 和 v_{r2} 不在 $p_0 q_{rk}$ 的同侧，这时候 q_{rk} 一定不是 SP(s，t)上的点。

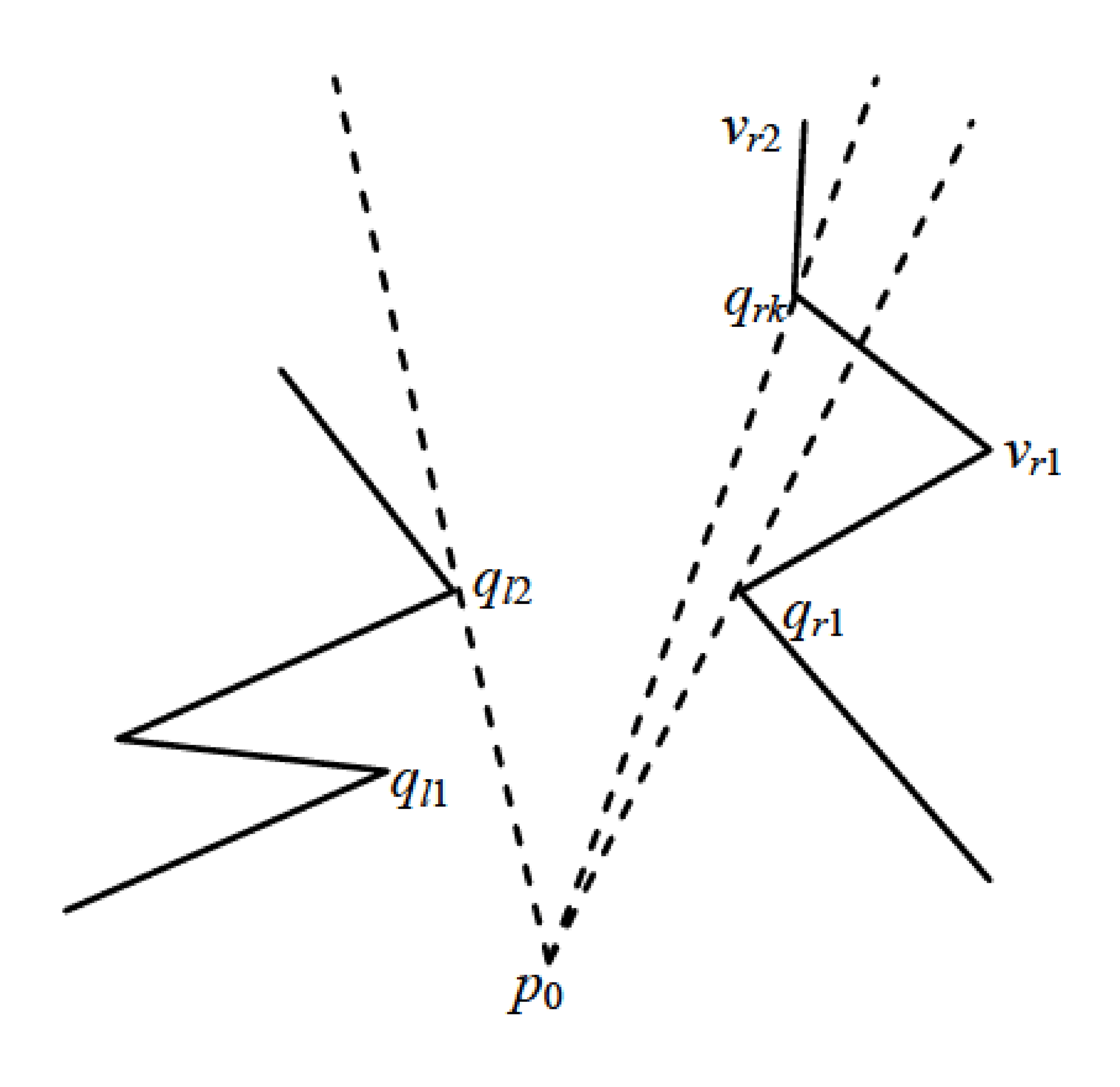


图 3.25　q_{rk} 在 $p_0 q_{r1}$ 左侧且在 $p_0 q_{l2}$ 右侧

重复上面的计算直至遇到终点 t。

下面考虑当计算遇到终点 t 时候的情况。

如果左、右凸包中没有内尖点了，则 t 处于 p_0q_{l2}、p_0q_{r1} 构成的视角之内，则 p_0、t 之间是可见的，计算结束（见图 3.26（a））。

否则，若左凸包不是空，且 t 在 p_0q_{l2} 的左侧，计算该凸包上的点与 t 形成的新凸包，新凸包上的点都是 SP(s，t)上的结点（见图 3.26（b））；如果右凸包不是空，且 t 在 p_0q_{r1} 的右侧，计算该凸包上的点与 t 形成的新凸包，新凸包上的点都是 SP(s，t)上的结点（见图 3.26（c））。整个计算结束。

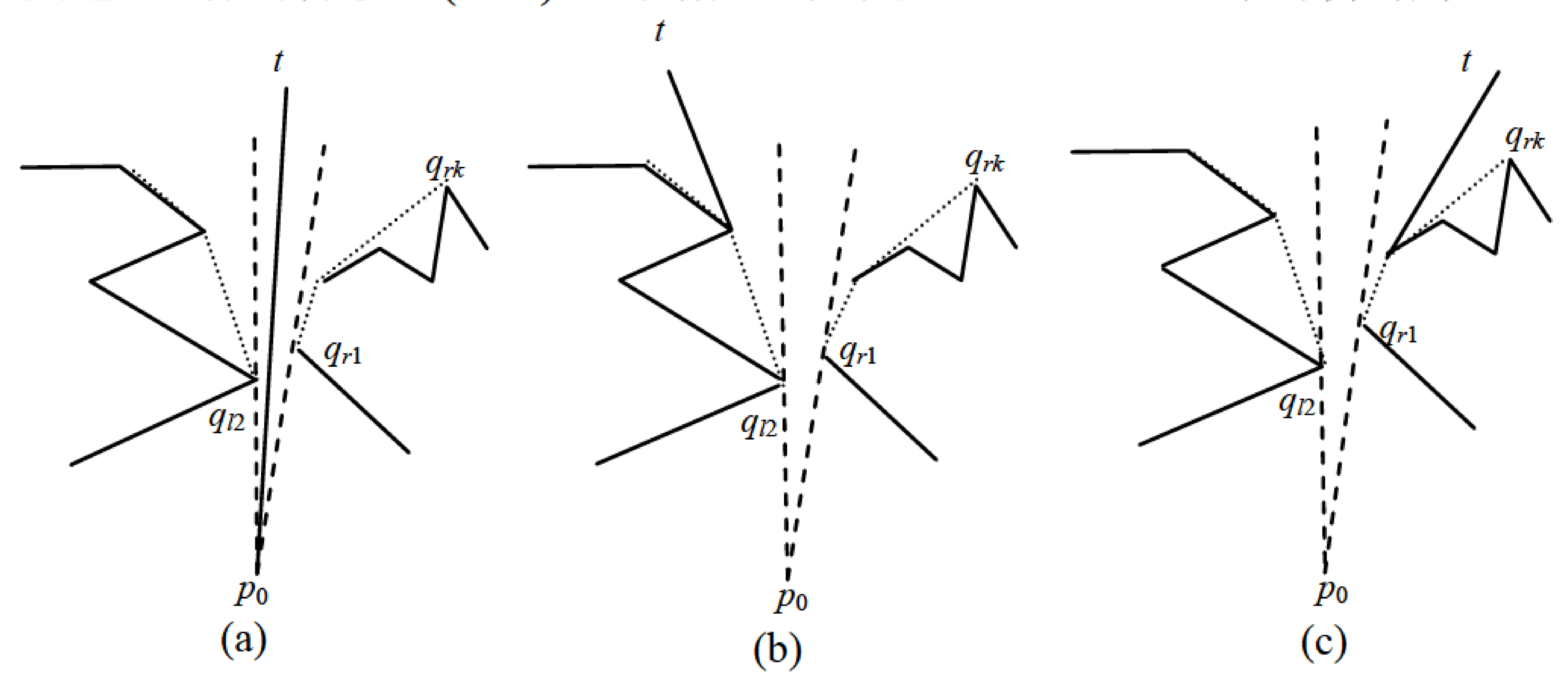


图 3.26　遇到 t 的三种情况

算法3.7　沿着Voronoi骨架路径VSP(s，t)计算最短路径SP(s，t)

令p是SP(s，t)上的当前顶点，q_k是VSP(s，t)上的当前关联内尖点，CH(q_l)是左关联内尖点q_l到q_k的左凸包链；CH(q_r)是右关联内尖点q_r到q_k的右凸包链，TL(q_l)是从p到CH(q_l)的左切线，切点为q_l，TL(q_r)是从p到CH(q_r)的右切线，切点为q_r。

（1）$p=s$；CH(q_l) = NULL；CH(q_r) = NULL；k =1；输出p。

（2）对于VSP(s，t)上的当前Voronoi边e_k，设$q_{k-1}q_k$是当前的检查边。

（2.1）如果$q_{k-1}q_k$是左检查边，则：

（2.1.1）如果$q_{k-1}q_k$与TL(q_r)相交，沿CH(q_r)移动p和q_r，输出p，转（2.1.1）；

（2.1.2）否则，重新计算CH(q_l)和TL(q_l)。

（2.2）如果$q_{k-1}q_k$是右检查边，则：

（2.2.1）如果$q_{k-1}q_k$与 TL(q_l)相交，沿CH(q_l)移动p和q_l，输出p，转（2.2.1）；

（2.2.2）否则，重新计算CH(q_r)和TL(q_r)。

（2.3）k++。

（3）如果CH(q_l)= NULL 且CH(q_r) = NULL，则输出t，返回。

（4）如果CH(q_l)!= NULL且t在TL(q_l)左侧，

（4.1）输出q_l;

（4.2）如果q_l是CH(q_l)的最后一个顶点，转（6);

（4.3）设CH(q_l)上q_l后面的点为a，令p=q_l，q_l=a；转（4）。

（5）如果CH(q_r) != NULL且t在TL(q_r)右侧，

（5.1）输出q_r;

（5.2）如果q_l是CH(q_r)的最后一个顶点，转（6）；

（5.3）设CH(q_r)上q_r后面的点为b，令p= q_r，q_r =b；转（5）。

（6）输出t；返回。

算法分析：简单多边形的 Voronoi 图的顶点个数为 $n+k-2$，边数为 $2(n+k)-3$，其中 n 是多边形的顶点数，$k(k<n)$是内尖点的数目，寻找骨架路径 VSP(s，t)的时间复杂度为 $O(n)$，在计算最短路径时是沿着 VSP(s，t)上关联内尖点顺序计算的，计算过程中每个关联内尖点的处理为常数次。所以通过 VSP(s，t)计算 SP(s，t)所用时间为 $O(n)$。由于找到起点与终点的步骤花费

的时间仅为 $O(n)$，所以该最短路径算法的总时间复杂度为 $O(n)$。

对于基于 Voronoi 图的复杂多边形中的最短路径计算，可参见文献[WangXT2011]。

3.3.4　复杂多边形中的可见性计算

如果多边形 P 内两点 p 和 q 之间的线段上没有多边形 P 外部的点，则 p 和 q 是相互可见的；多边形 P 中某点 p 的可见多边形 VP(p)是指 P 中所有从 p 看去可见的点的集合。计算多边形内一点 p 的可见多边形，即找到多边形中相对于 p 点可见的全部点的集合，是一系列多边形可见性计算问题中最基本的一个问题。

本节介绍一种基于多边形 Voronoi 图的点可见性算法[Zhao2013]。算法可以在“带洞”多边形中应用，并只在所给查询点的可见多边形周围的局部范围内进行遍历。算法数据结构比较简单，剖分空间合理且易于实现，只需要 $O(n)$空间和 $O(n\log n)$预处理时间。应用实例与测试分析表明，该算法是一种实际可行的算法，且运行时间与|VP(v)|和 n 均呈线性关系，其中|VP(v)|为点 v 的可见多边形的边数，n 为多边形的边数。

本节规定，多边形的外边界是顺时针的，内边界是逆时针的。我们将多边形中的凹顶点看作特殊的边，即它的两个端点是重合的。这样，简单多边形的 Voronoi 图是一棵树，其叶结点是多边形的顶点；“带洞”多边形的 Voronoi 图是一个图，其中度数为 1 的结点为多边形的顶点。

1. 概念与性质

VP(v)有两种类型的边：一类是原始边，是P的边或边的一部分，称其为P的边相对于v的可见部分；另一类为构造边，其端点在P的边界上，除端点外的部分则在P的内部。对于简单多边形，每条边相对于v最多只有一个可见部分；对于“带洞”多边形，每条边相对于v可能有多个可见部分。由于VP(v)是一个星形多边形，因此如果逆时针沿着VP(v)的边界依次遍历，v和VP(v)的各顶点的连线与水平轴之间的有向角是从小到大依次排列的。算法将沿着多边

形的Voronoi图进行深度优先搜索，访问多边形中的边，并计算被访问边的相对于v的可见部分，其中会用到v到其两个顶点的Voronoi骨架路径和局部最短路径的概念。

对于多边形 P 中一点 v，假设其位于边 e 的 Voronoi 区域 VR(e)中，我们过 v 作 e 的垂线，可以证明该垂线只与 VR(e)的边界上的一条 Voronoi 边相交，不妨设交点为 a。我们将 a 看作一个结点，va 看作一条边加入 VD(P)（如图 3.27 所示）。

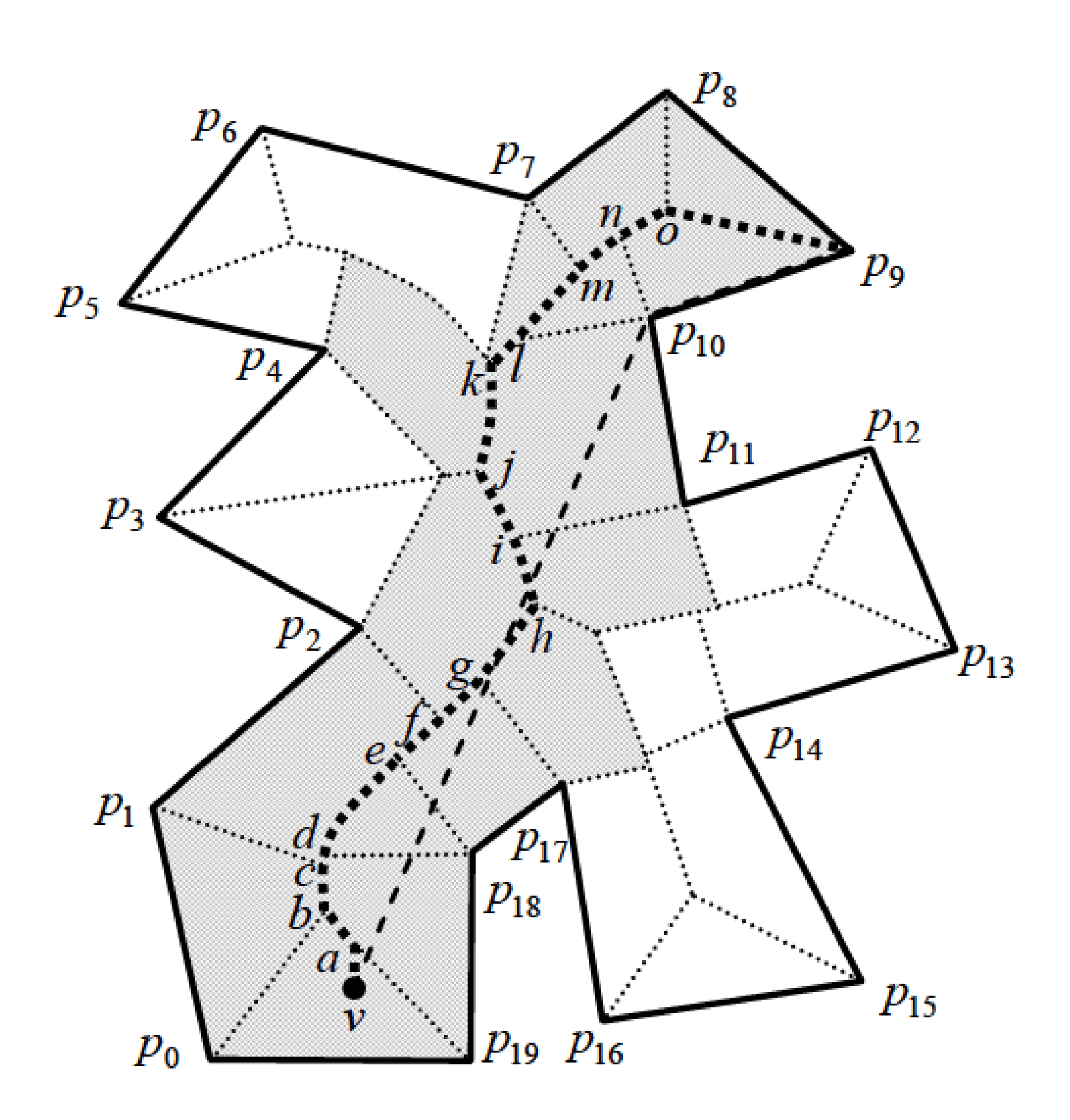


图 3.27　简单多边形中 Voronoi 骨架路径（粗虚线）和 Voronoi 通道（灰区域）

由 3.3.3 节可知，v 到 P 的一个顶点 p_i 的 Voronoi 骨架路径 VSP(v，p_i)是一条由系列 Voronoi 边首尾依次连接而成的路径。如图 3.27 中粗虚线所示，VSP(v，p_9)= {v，a，…，o，p_9}。边 $p_1 p_2$ 为有向边 de 的左侧关联边，凹顶点（特殊边）p_{18} 为有向边 de 的右侧关联边。需要注意的是，$p_1 p_2$ 为有向边 ed 的右侧关联边，p_{18} 为有向边 ed 的左侧关联边。

多边形 P 中的一个点 v 到 P 的一个顶点 p_i 的 Voronoi 骨架路径 VSP(v，p_i)的每条 Voronoi 边所关联的 Voronoi 区域形成 v 到 p_i 的一个通道，称之为 VSP(v，p_i)对应的 Voronoi 通道 C(v，p_i)。v 到两个顶点 p_i 、p_{i+1} 的 Voronoi 通道 C(v，p_i)和 C(v，p_{i+1})组成了 v 到边 $\overline{p_i p_{i+1}}$ 的 Voronoi 通道 C(v，$p_i\, p_{i+1}$)。

这里，VSP(v，p_i)和 VSP(v，p_{i+1})是在沿 VD(P)深度优先搜索时依次遍历得到的，它们除了所含有的 VR($p_i p_{i+1}$)的 Voronoi 边不同，其他边都是相同的。

在一个 C(v，p_i)中，将 VSP(v，p_i) 看作一条绳子，如果用力将其拉紧，则可得到在该 C(v，p_i)中 v 到 p_i 的一条最短路径，我们将其称之为局部最短路径，用 SP$_c$(v，p_i)表示。

图 3.27 中，灰色区域为 VSP(v，p_9)所对应的 Voronoi 通道 C(v，p_9)，图 3.28 中灰色区域为 v 到 p_3 的两条 Voronoi 骨架路径分别对应的 Voronoi 通道。图 3.27 中，VSP(v，p_8)= {v，a，…，o，p_8}，VSP(v，p_9)= {v，a，…，o，p_9}，它们不同的部分是 op_8 和 op_9，都是 VR($p_8 p_9$)的 Voronoi 边。在 C(v，p_8p_9)中，v 到 p_8 的局部最短路径为 SP$_c$(v，p_8)={v，p_8}，v 到 p_9 的局部最短路径为 SP$_c$(v，p_9)={v，p_{10}，p_9}。需要注意的是，在“带洞”多边形中，v 到 p_i 存在多个 Voronoi 通道，因此存在多个局部最短路径。如在图 3.28（a）中的 Voronoi 通道 C(v，p_3)（灰色区域）中，SP$_c$(v，p_3)={v，p_3}；在图 3.28（b）中，还存在一个 Voronoi 通道（灰色区域），其中 SP$_c$(v，p_3)={v，p_{11}，p_{13}，p_3}。

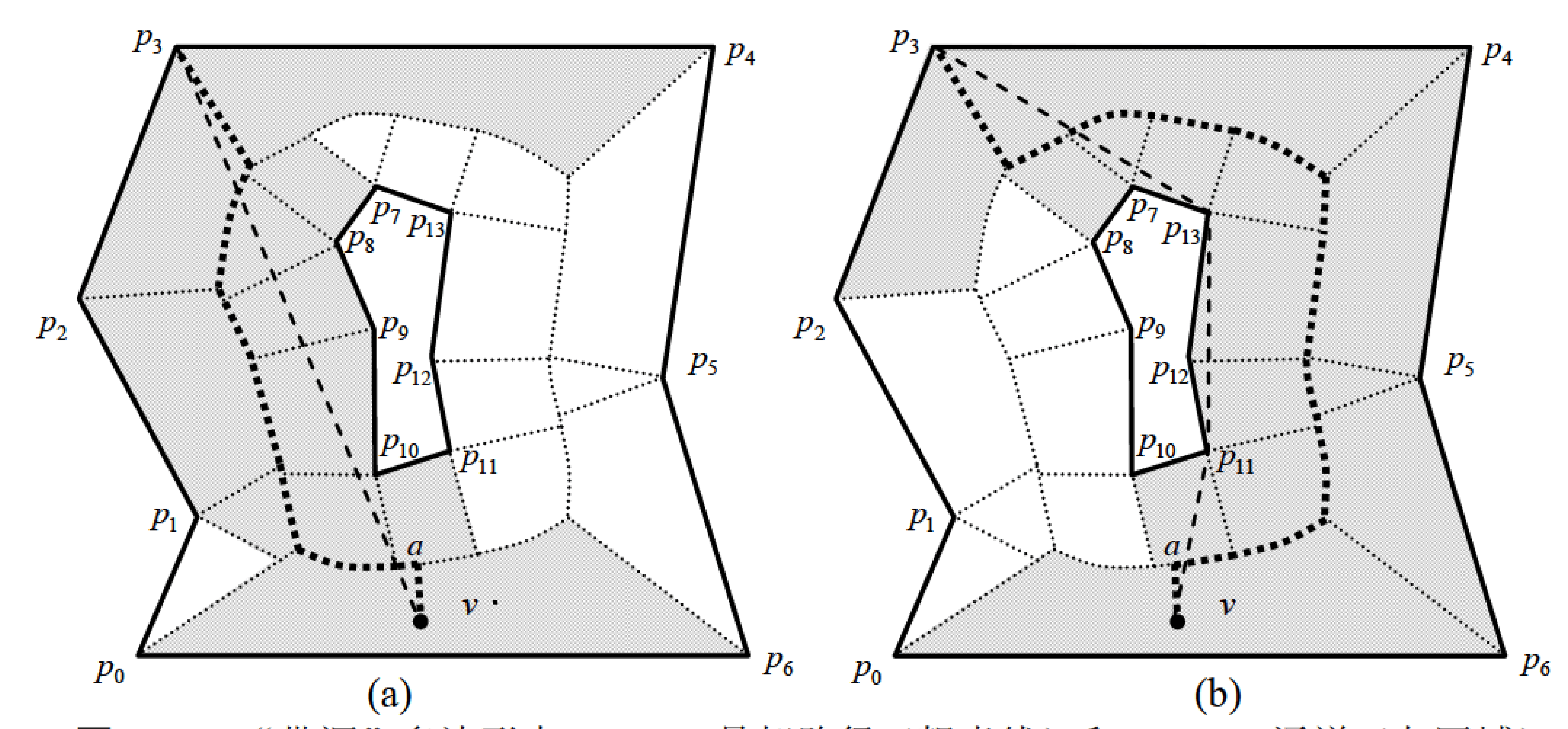


图 3.28　“带洞”多边形中 Voronoi 骨架路径（粗虚线）和 Voronoi 通道（灰区域）

2. 算法描述

本节首先讨论多边形内计算一条边相对于给定查询点的可见部分的计算方法。

由于对于一个多边形 P 及其内部一点 v，v 到 P 的顶点 p_i 的一个 Voronoi 通道可以看作一个简单多边形，由定理 3.13 可得引理 3.12。

引理 3.12 对于一个多边形 P 及其内部一点 v，在 v 到 p_i 的一个 Voronoi 通道中，v 到 p_i 的局部最短路径 $SP_c(v, p_i)$能够通过顺序遍历 $VSP(v, p_i)$得到，且 $SP_c(v, p_i)$上的顶点必为 $VSP(v, p_i)$所关联的内尖点。

引理 3.13 对于一个多边形 P 及其内部一点 v，在 v 到 P 的边 p_ip_{i+1} 的一个 Voronoi 通道中，p_ip_{i+1} 相对于 v 存在可见部分，当且仅当 v 到 p_i 的局部最短路径 $SP_c(v, p_i)$是一条在 P 内的凸多边形链且在 vp_i 的右侧，且 v 到 p_{i+1} 的局部最短路径 $SP_c(v, p_{i+1})$也是一条在 P 内的凸多边形链且在 vp_{i+1} 的左侧。

证明：充分性：由引理 3.12 知，$SP_c(v, p_i)$上的点为 $VSP(v, p_i)$所关联的内尖点。如果只用 $VSP(v, p_i)$的左侧关联内尖点计算 $SP_c(v, p_i)$，则 $SP_c(v, p_i)$为 $VSP(v, p_i)$的左侧关联内尖点的凸多边形链 L_CP。在 p_ip_{i+1} 相对于 v 存在可见部分的情况下，若 $SP_c(v, p_i)\neq$L_CP，即 $SP_c(v, p_i)$非凸，则存在 $SP_c(v, p_i)=\{v, l_1, \ldots, l_l, p_i\}$中一个顶点 $l_j(1\leqslant j\leqslant l)$为 $VSP(v, p_i)$的右侧关联内尖点，则 $SP_c(v, p_i)$和 $SP_c(v, p_{i+1})$有交错部分，p_ip_{i+1} 相对于 v 不可见，存在矛盾。因此，$SP_c(v, p_i)=$ L_CP，即 $SP_c(v, p_i)$是一条在 P 内的凸多边形链。显然，该链在 vp_i 的右侧。同理可证 $SP_c(v, p_{i+1})$也是一条在 P 内的凸多边形链且在 vp_{i+1} 的左侧。

必要性：设 $SP_c(v, p_i)=\{v, l_1, \cdots, l_l, p_i\}$，$SP_c(v, p_{i+1})=\{v, r_1, \cdots, r_r, p_{i+1}\}$，显然，$SP_c(v, p_i)$在 $VSP(v, p_i)$的左侧，$SP_c(v, p_{i+1})$在 $VSP(v, p_{i+1})$的右侧。由于 $VSP(v, p_i)$在 $VSP(v, p_{i+1})$的左侧，因此 $SP_c(v, p_i)$在 $SP_c(v, p_{i+1})$的左侧。由于 $SP_c(v, p_i)$是一条凸多边形链且在 vp_i 的右侧，且 $SP_c(v, p_{i+1})$也是一条凸多边形链且在 vp_{i+1} 的左侧，它们形成一个漏斗形状，因此 $p_i p_{i+1}$ 相对于点 v 有可见部分。证毕。

如图 3.29 所示，对于边 $p_4 p_5$，有局部最短路径 $SP_c(v, p_4)=\{vp_3 p_4\}$，$SP_c(v, p_5)=\{vp_6p_5\}$，符合引理 3.13 的条件，可以断定 p_4p_5 相对 v 有可见部分。而对于边 p_3p_4，有局部最短路径 $SP_c(v, p_3)=\{v p_3\}$，$SP_c(v, p_4)=\{vp_3 p_4\}$，

不符合引理 3.13 的条件，所以可以断定 p_3p_4 相对于 v 不可见（严格地说，点 p_3 相对于 v 可见，在本文中定义类似 p_3p_4 这样的边不可见，不会影响算法输出结果）。

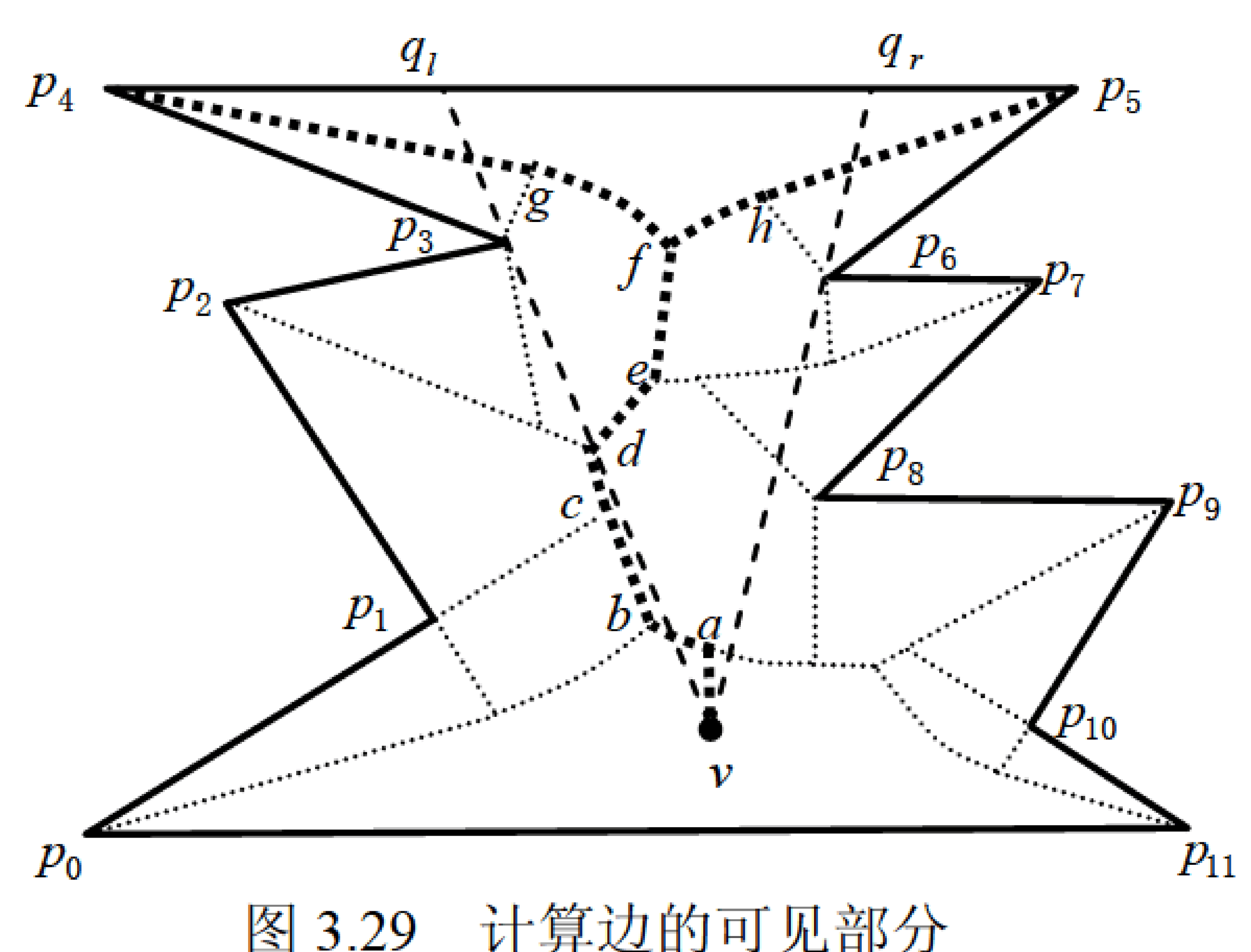


图 3.29　计算边的可见部分

由引理 3.12 和引理 3.13 容易得到定理 3.14。

定理 3.14　对于一个多边形 P 及其内部一点 v，在 v 到 P 的边 $p_i\,p_{i+1}$ 的一个 Voronoi 通道中，有 $SP_c(v，p_i)=\{v，l_1，…，l_l，p_i\}$，$SP_c(v，p_{i+1})=\{v，r_1，\cdots，r_r，p_{i+1}\}$。$p_i\,p_{i+1}$ 相对于 v 存在可见部分，当且仅当 vl_1 在 vr_1 的左侧，而且当前 Voronoi 通道中 $p_i\,p_{i+1}$ 相对于 v 的可见部分可通过计算射线 vl_1 和 vr_1 与 $p_i\,p_{i+1}$ 的交点得到。

如图 3.29 中，vp_3 在 vp_6 的左侧，根据定理 3.14 知，$p_4\,p_5$ 相对于 v 存在可见部分。通过计算射线 vp_3、vp_6 与 p_4p_5 的交点即可计算出 p_4p_5 相对于 v 的可见部分为 $q_l\,q_r$。这里，射线 vl_1 和 vr_1 之间的区域成椎体状，称为可见椎体，用 $VC(vl_1，vr_1)$表示。

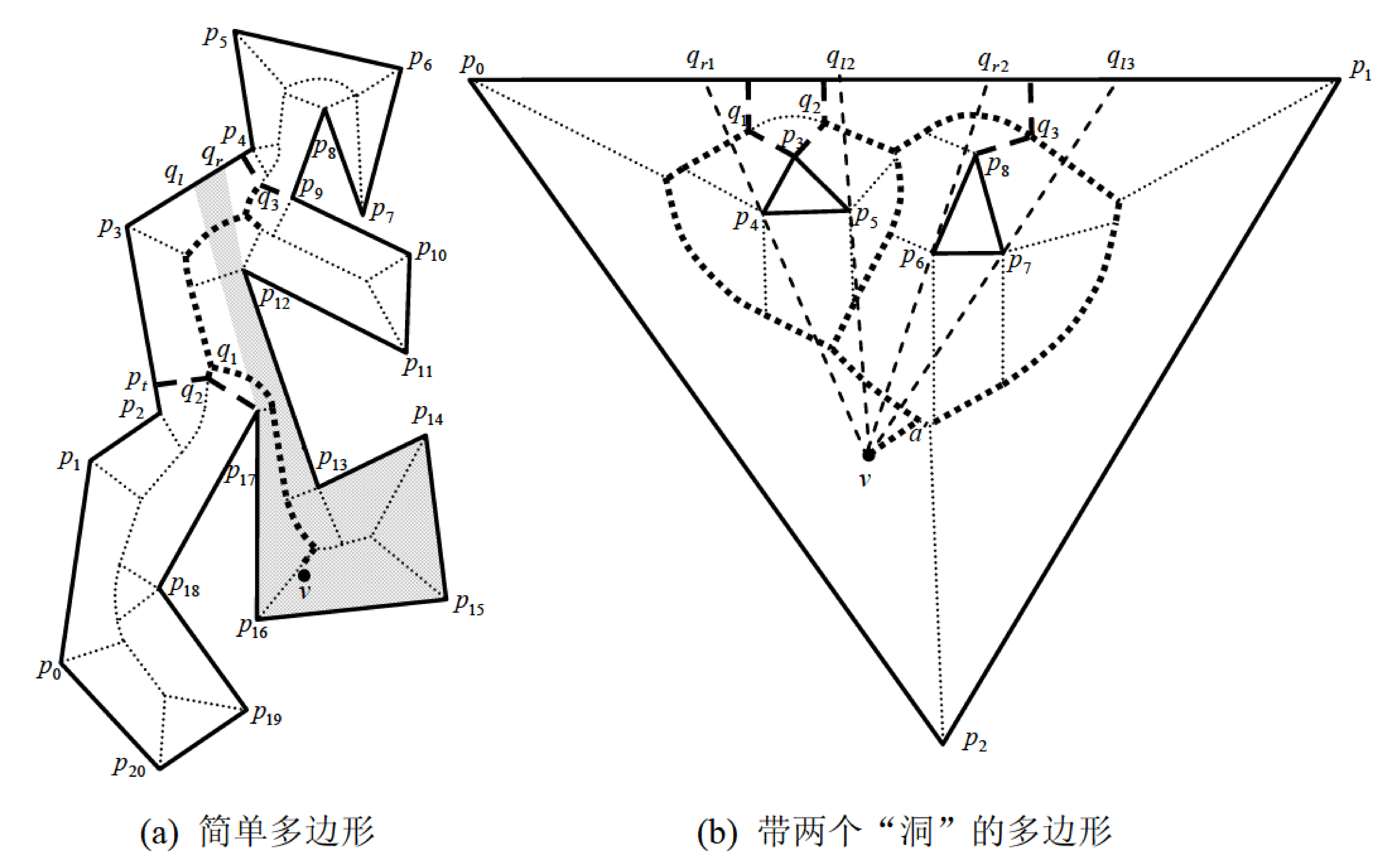
为了计算整个 VP(v)，我们将沿着 VD(P)进行深度优先搜索，访问每个叶结点，即访问 P 中的每个顶点。由于访问中得到的相邻两个顶点属于同一条边，所以可以在前一顶点的 Voronoi 骨架路径基础上，增加或减少新访问的 Voronoi 边，即可得到下一顶点的 Voronoi 骨架路径；并在计算新的 Voronoi 骨架路径的同时，在前一顶点的局部最短路径的基础上计算下一顶点的局部最短路径。这样可减少计算量。

在沿着 VD(P)进行深度优先搜索时，需要考虑每次遍历的终止条件，原因在于以下两点。

（1）在图 3.30（a）中，VP(v)（灰色区域）只是多边形的一部分，在遍历到 Voronoi 顶点 q_2 和 q_3 时，都没有必要再深度搜索下去，因为它们后面的 Voronoi 边关联的多边形的边肯定相对于 v 不可见。

（2）在图 3.30（b）中的“带洞”多边形，如果不设终止条件，会陷入死循环。其实，在沿每个“洞”的一侧遍历时，当遍历到某个 Voronoi 顶点（如 q_1、q_2 和 q_3）后，后面的 Voronoi 边对应的边都相对于 v 不可见，所以可以停止这个方向的遍历。

下面讨论如何设定终止条件。很容易错误地认为：根据定理 3.14，如果 vl_1 不在 vr_1 的左侧，则可以停止继续往下遍历并开始回溯。但在图 3.31 中，在沿“洞”H_1 左侧遍历到顶点 p_3 时，vl_1=vp_1 不在 vr_1=vp_1 的左侧，但仍然需要继续遍历，并经过点 q_3、q_2、q_4，直至到达 q_6 才能停止。否则，多边形的某些边（如 p_4p_5、p_5p_6）相对于 v 的可见部分就会漏掉。



(a) 简单多边形　　(b) 带两个“洞”的多边形

图 3.30　终止条件示例（粗虚线）

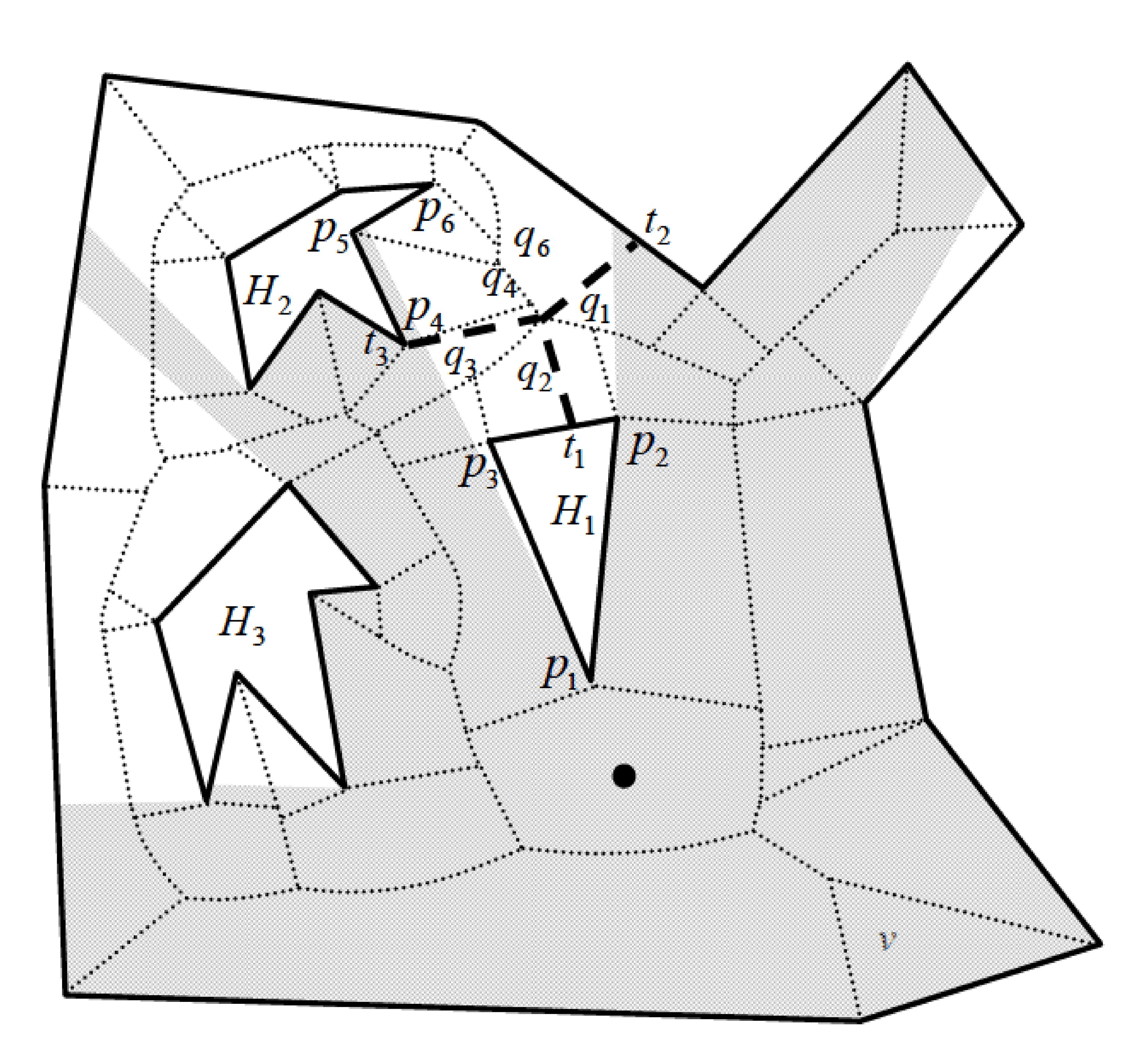


图 3.31　遍历方向不同，q_2 处的终止条件计算方法不同

不妨设当前遍历的骨架路径为 v，a，a_1，a_2，…，a_j，当前的可见椎体为 VC(vl_1, vr_1)，进行深度搜索访问到的下一个 Voronoi 顶点为 a_{j+1}。过 a_{j+1} 分别作 a_ja_{j+1} 的两侧关联边的垂足 t_1 和 t_2（需要注意的是，若另一个遍历是从 a_{j+1} 遍历到 a_j，应当过点 a_j 作 Voronoi 边 $a_{j+1}a_j$ 左右两侧关联的边的垂足。如图 3.31 所示，当从点 q_3 遍历到 q_2，取垂足为 t_1 和 t_3，若从 q_1 遍历到 q_2，垂足为 t_1 和 t_2，若从 q_4 遍历到 q_2，垂足为 t_3 和 t_2）。

定理 3.15　在当前访问 Voronoi 通道中，若点 t_1 和 t_2 同时在 vl_1 的左侧或在 vl_1 上，或者同时在 vr_1 的右侧或在 vr_1 上，则沿着 VD(P)进行的当前遍历中，a_{j+1} 后面的 Voronoi 边 e 所关联的边都不在 VC(vl_1，vr_1)中。

证明：假设命题不成立。则在当前访问的 Voronoi 通道中，假设 a_{j+1} 后面的 Voronoi 边 e 所关联的边上有一点 p 在 VC(vl_1，vr_1)中，即 p 相对于 v 可见，则有 SP$_c$(v，p)= (v，p)。然而，由于 t_1 和 t_2 同时在 vl_1 左侧或在 vl_1 上，或者同时在 vr_1 右侧或在 vr_1 上，e 为沿着 VD(P)在 a_{j+1} 后遍历到的 Voronoi 边所关联的边，所以 p 必在远离 VC(vl_1，vr_1)的一侧，因而可知 l_1 或 r_1 必为 v 到 p 的局部最短路径上一点，这与 SP$_c$(v，p)= (v，p)矛盾，定理 3.15 得证。

证毕。

定理 3.15 给出了沿 VD(P)进行深度遍历的终止条件。我们根据定理 3.15 设计函数 TC(q)，用于表示在当前访问的 Voronoi 通道中 Voronoi 顶点 q 是否满足终止条件。如果 TC(q)=True，则满足终止条件。如图 3.30（a）所示，在当前访问的 Voronoi 通道中，p_{17} 和 p_t 为点 q_2 在边 q_1q_2 的两侧关联边的垂足，p_{17} 在 vp_{17} 上，p_t 在 vp_{17} 的左侧，因此 TC(q_2)=True。该多边形中，在当前访问的 Voronoi 通道中，多边形 $p_tp_2p_1p_0p_{20}p_{19}p_{18}p_{17}$ 内的点相对于视点 v 为不可见。同理，在相应访问的 Voronoi 通道中，TC(q_3)为真。在图 3.30（b）中，在相应访问的 Voronoi 通道（粗虚线为相应的 Voronoi 骨架）中，TC(q_1)、TC(q_2)和 TC(q_3)为真，其中点 q_3 是从不同的遍历方向（不同方向所访问的 Voronoi 通道）判断了两次终止条件。粗虚线为搜索过程中遍历到的 Voronoi 边。在图 3.31 中，当从 H_1 的左侧向右侧遍历（从 q_3 朝 q_2 和 q_1 方向遍历）时，遍历到 q_3 时，TC(q_3)为假；访问到 q_2 时，用到垂足 t_3 和 t_1，因为点 t_3 相对于点 v 可见，TC(q_2)为假；只有遍历到 q_1 时，有 TC(q_1)为真，此次遍历才终止。当从 H_1 的右侧向左侧遍历（从 q_1 朝 q_2 和 q_3 方向遍历）时，遍历到 q_1 时，TC(q_1)为假；遍历到 q_2 时，用到垂足 t_2 和 t_1，因为点 t_2 对于点 v 可见，TC(q_2)同样为假；只有遍历到 q_3 时有 TC(q_3)为真，此次遍历才终止。

基于上面的描述，算法 3.8 给出了计算点的可见多边形的算法。这里假设给定一个有 n 条边和 $h(\geqslant 0)$个“洞”的多边形 P 以及 P 中一点 v，在预处理阶段已计算出 P 的 Voronoi 图 VD(P)。

算法 3.8　计算点的可见多边形

（1）判断 v 所在的 Voronoi 区域 VR(o)，假设 $o=p_ip_{i+1}$；

（2）计算 VSP(v，p_i)和 VSP(v，p_{i+1})，同时计算 SP$_c$(v，p_i)和 SP$_c$(v，p_{i+1})；

（3）计算 o 对于 v 的可见部分；

（4）取 SP$_c$(v，p_{i+1})最后两个点 q_{r-1}、q_r；//q_r 为多边形的顶点；

（5）如果 TC(q_{r-1})=False

（5.1）取 $q_{r-1}q_r$ 右侧关联边，作为新的 p_ip_{i+1}；

（5.2）如果 $p_ip_{i+1}=o$，算法结束；

（5.3）计算 VSP(v，p_i)和 VSP(v，p_{i+1})，同时计算 SP_c(v，p_i)和 SP_c(v，p_{i+1})；

（6）否则，

（6.1）取 $q_{r-2}q_{r-1}$ 右侧关联边，作为新的 p_ip_{i+1}；

（6.2）计算 VSP(v，p_i)和 VSP(v，p_{i+1})，同时计算 SP_c(v，p_i)和 SP_c(v，p_{i+1})；

（7）转（3）。

需要说明的是，算法 3.8 中的第（2）、（5.3）和（6.2）步中计算 VSP(v，p_i)和 VSP(v，p_{i+1})的方法不同。下面将分别对它们进行讨论。

3. Voronoi 骨架路径和局部最短路径的计算

算法 3.8 的第（1）步首先确定给定点 v 所在的 Voronoi 区域 VR(o)。我们可通过逐个测试每个 Voronoi 区域来实现，需要 $O(n)$时间。算法 3.8 的第（2）步用于计算初始的 Voronoi 最短路径和局部最短路径。在加入了点 a 和边 va 的 VD(P)中，令 VSP(v，p_i)={v，a，q_k，…，q_0，p_i}，VSP(v，p_{i+1})={v，a，q_{k+1}，…，q_j，p_{i+1}}，其中 q_0，q_1，…，q_j 为 VR(o)的 Voronoi 顶点，a 在 q_k 和 q_{k+1} 之间。采用类似 3.3.3 节中的方法，在遍历 VSP(v，p_i)(VSP(v，p_{i+1}))的同时，找到被访问 Voronoi 边的两侧的关联凹顶点，逐次加入到已生成的局部最短路径中，修改生成新的局部最短路径，最后可得到 SP_c(v，p_i)(SP_c(v，p_{i+1}))。如图 3.32 所示，在加入了点 a 和边 va 的 VD(P)中，VSP(v，p_{13})={v，a，q_8，q_7，…，q_0，p_{13}}，VSP(v，p_0)={v，a，q_9，q_{10}，…，q_{14}，p_0}。在遍历 VSP(v，p_{13})(VSP(v，p_0))的同时，计算局部最短路径，最后可得到 SP_c(v，p_{13})={v，p_7，p_9，p_{13}}(SP_c(v，p_0)={v，p_4，p_2，p_0})。由于 vp_7 在 vp_4 的左侧，因此边 p_0p_{13} 相对于点 v 的可见部分为 q_rq_l，其中 q_rq_l 为 vp_7、vp_4 与 p_0p_{13} 的交。

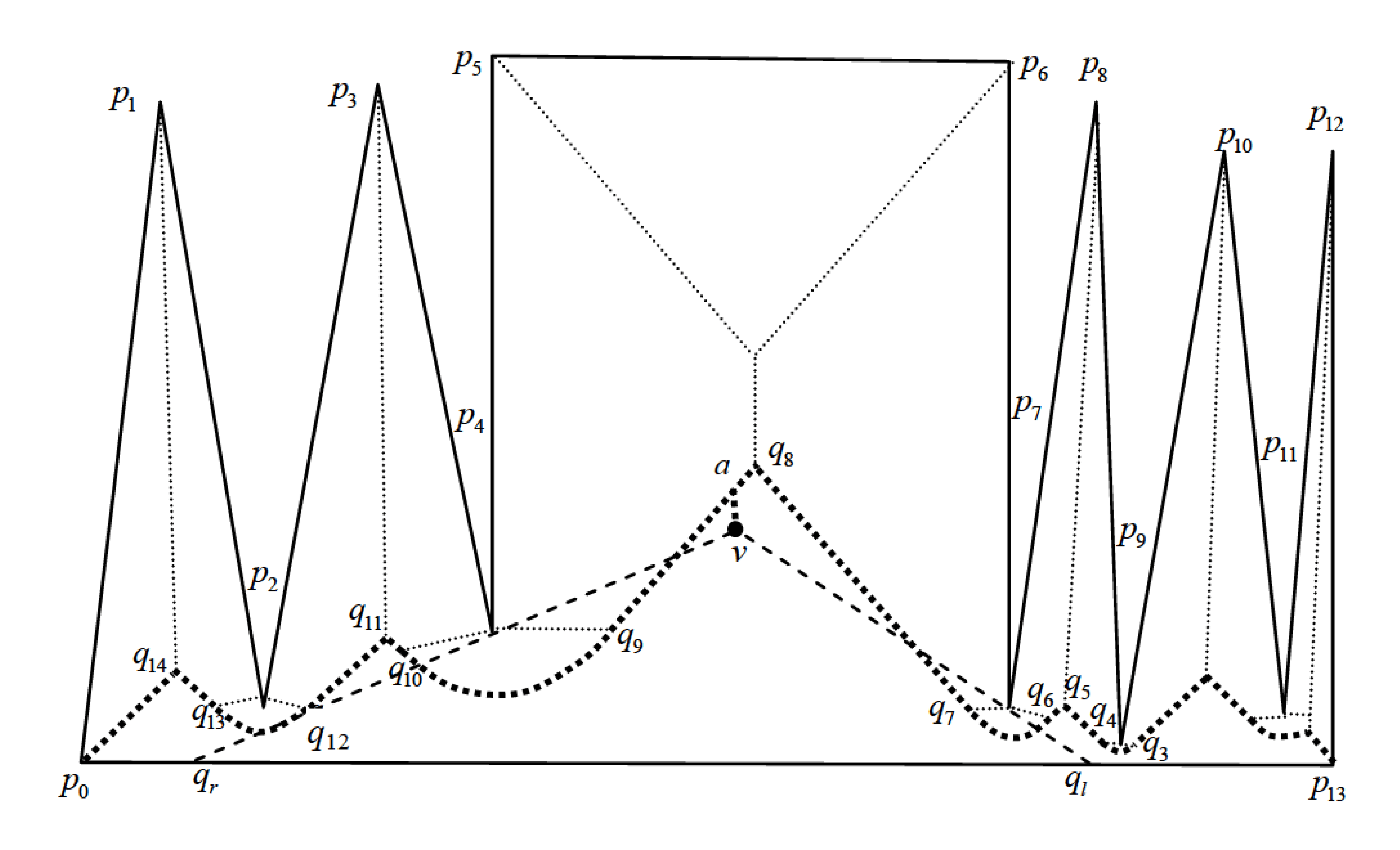


图 3.32 计算初始 Voronoi 骨架路径

我们注意到，在从 a 到 p_{13} 的访问过程中，当访问到 Voronoi 顶点 q_4 时，TC(q_4)为真，这时没必要继续访问 q_3、q_2、q_1 和 q_0。这时得到的局部最短路径的第一条边 vp_7 就是最后用于计算 $p_{13}p_0$ 相对于 v 的可见部分的那条射线。不难证明，即使遍历完 q_3、q_2、q_1 和 q_0，所得到的 v 到 p_{13} 的最短路径的第一条边仍为 vp_7，因为 q_3，q_2，q_1 和 q_0 在 vp_7 左侧。因此，这里可以测试终止条件，减少计算。

在已计算 VSP(v，p_{i+1})和 SP$_c$(v，p_{i+1})的基础上，考虑如何快速确定下一条边对应的两条 Voronoi 骨架路径和局部最短路径。这取决于 TC(q_{r-1})的值，如下所述。

（1）TC(q_{r-1})=False 时，可以继续往前遍历。这时，取 SP$_c$(v，p_{i+1})中的最后两个点 q_{r-1}、q_r。我们知道 q_r 为多边形的顶点 p_{i+1}。此时，Voronoi 边 $q_{r-1}q_r$ 的右侧关联边为 $p_{i+1}\ p_{i+2}$。这时，VSP(v，p_{i+1})和 SP$_c$(v，p_{i+1})已知，只须计算 VSP(v，p_{i+2})和 SP$_c$(v，p_{i+2})。令 VSP(v，p_{i+2})= VSP(v，p_{i+1})，SP$_c$(v，p_{i+2})= SP$_c$(v，p_{i+1})。对 VD(P)进行遍历，直到到达 p_{i+2} 对应的叶结点，同时对于这个过程中所遍历的 Voronoi 边，如果该边原来被遍历过，则其已在 VSP(v，

p_{i+2})中，需要将其从 VSP(v，p_{i+2})中删除；否则，将其追加到 VSP(v，p_{i+2})中，同时修改 SP_c(v，p_{i+2})。

在图 3.29 中，已知 VSP(v，p_4)={$vabcdefgp_4$}，需要在其基础上计算 VSP(v，p_5)。VSP(v，p_5)初始值为{$vabcdefgp_4$}。在对 VD(P)进行遍历到达 p_5 对应的叶结点过程中，需要从 VSP(v，p_5)中删除 gp_4 和 fg，加入 fh 和 hp_5，得到 VSP(v，p_5)={$vabcdef hp_5$}。在这个过程中，同时修改 SP_c(v，p_5)，最后得到 SP_c(v，p_5)={vp_6p_5}。

类似前面计算初始 Voronoi 骨架路径所做的分析和处理，在朝 p_{i+2} 遍历的过程中，同时进行终止条件的测试，遇到满足终止条件的情况就停止继续往前遍历。这时得到的局部最短路径的第一条边可用于边的可见部分的计算。遇到这种情况时，计算完边的可见部分后，须转算法 3.8 的第（6）步。

（2）TC(q_{r-1})=True 时，不用继续往前遍历。这时，取 SP_c(v，p_{i+1})中的最后三个点 q_{r-2}、q_{r-1}、q_r。我们知道 q_r 为多边形的顶点 p_{i+1}。此时，设 Voronoi 边 $q_{r-2}q_{r-1}$ 的右侧关联边为 p_jp_{j+1}。这时做如下处理。

① 令初始 VSP(v，p_j)为 VSP(v，p_{i+1})，初始 SP_c(v，p_j)为 SP_c(v，p_{i+1})。沿 VR(p_jp_{j+1})朝 p_j 方向遍历，将遍历的 Voronoi 边加入 VSP(v，p_j)，并同时计算 SP_c(v，p_j)；

② 令初始 VSP(v，p_{j+1})为 VSP(v，p_{i+1})，初始 SP_c(v，p_{j+1})为 SP_c(v，p_{i+1})。沿 VR(p_jp_{j+1})朝 p_{j+1} 方向遍历，将遍历的 Voronoi 边从 VSP(v，p_{j+1})中删除或加入，并同时计算 SP_c(v，p_{j+1})。

然后令 $i=j$，继续新的处理。上述处理过程中，也都可以考虑进行终止条件的判断。

在图 3.30（a）中，算法会在 q_2 处停止继续往前遍历，将转向处理边 p_2p_3。用上面方法计算得到 SP_c(v，p_2)={v，p_{17}，p_2}，SP_c(v，p_3)={v，p_{17}，p_3}，可知 p_2p_3 相对于 v 不可见。算法也会在 q_3 处停止继续往前遍历，将转向处理凹顶点 p_9 开始回溯。在图 3.30（b）中，算法会在 q_1 处停止继续往前遍历，将转向处理边 p_3p_4。用上面方法计算得到 SP_c(v，p_4)={v，p_4}，SP_c(v，p_3)={v，

p_4，p_3}，可知 p_3p_4 相对于 v 不可见。

3.3.5 虚拟室内场景设计与漫游系统

虚拟室内场景可以看作是将二维平面多边形垂直拉伸后得到三维墙体，然后再在其中布置物品。虚拟博物馆设计与漫游系统即为其中一种具体应用[WangL2006]。其场馆的基本数据结构是 2D 多边形及其 Voronoi 图，它们是协同设计、馆内物品组织、可见性计算、碰撞检测、路径规划等的基础。建立 3D 博物馆场景时，使用的方法是将 2D 多边形进行高度拉伸获得 3D 博物馆外墙；通过 2D 多边形的 Voronoi 图计算多边形的 Offset 曲线，偏移的距离由用户交互给出，将 Offset 曲线进行一定高度的拉伸，可以作为沿墙而设的展台；用户可以协同地往不同的 Voronoi 区域中放置各种形式的数字展品；场景中的数字展览品和虚拟替身通过 Voronoi 区域进行定位与管理；基于多边形的 Voronoi 图设计路径规划、可见性计算、碰撞检测等算法，用于加速漫游。图 3.33 显示了多边形及其 Voronoi 图的应用框架。本章介绍的多边形 Voronoi 图及其应用算法可用于此系统。

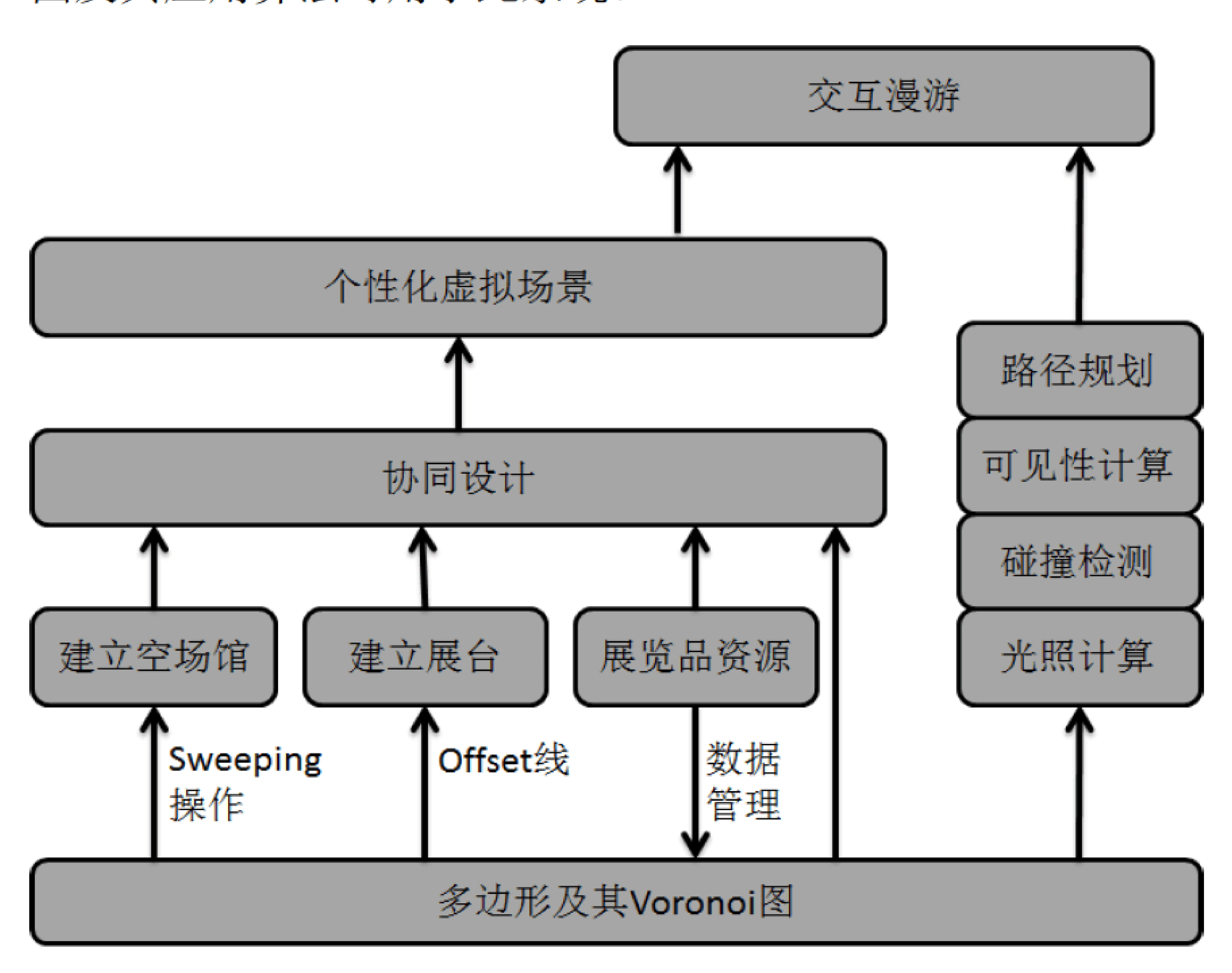


图 3.33　多边形及其 Voronoi 图的应用框架

第 4 章　约束 Delaunay 三角剖分及其应用

本章介绍平面直线图的三角剖分，特别是约束 Delaunay 三角剖分，包括其概念、性质、构造方法及若干应用。

4.1　定义与性质

由第 1 章知，在平面离散点集中添加边可将离散点集扩展为平面直线图，形成开放或封闭的平面区域。我们用 G=(P，E)表示一个平面直线图，其中 E 表示约束边的集合，P 表示离散点的集合。这里，P 也包含 E 中边的端点。

定义 4.1　给定平面直线图 G=(P，E)，对点集 P 进行一般的三角剖分，该剖分要求保证 E 中的边必须是该三角剖分中的边，称这种三角剖分为 G 的约束三角剖分（Constrained Triangulation），并称 E 中的边为约束边。

在实际应用中，约束边可以是河流、道路、山脊、海岸线等。

定义 4.2　如果 P 中两点之间的线段与 E 中任何约束边的内部都不相交，称这两个点是相互可见的；否则，该约束边对这两点产生了遮挡，称这两点是相互不可见的。

如图 4.1 所示的 G=(P，E)中，P={p_1，p_2，…，p_7}，E={p_1p_3}，p_1p_3 是约束边。由于其遮挡的缘故，p_4 相对于 p_2 不可见。在这个例子中，尽管 p_4 在三角形△$p_1p_2p_3$ 的外接圆（虚线）中，但由于 p_4 相对于 p_2 不可见，我们仍称△$p_1p_2p_3$ 的外接圆是一个空圆，这里称之为约束空圆。在三角剖分中构造这类空圆时所采用的准则，称之为约束空圆准则。

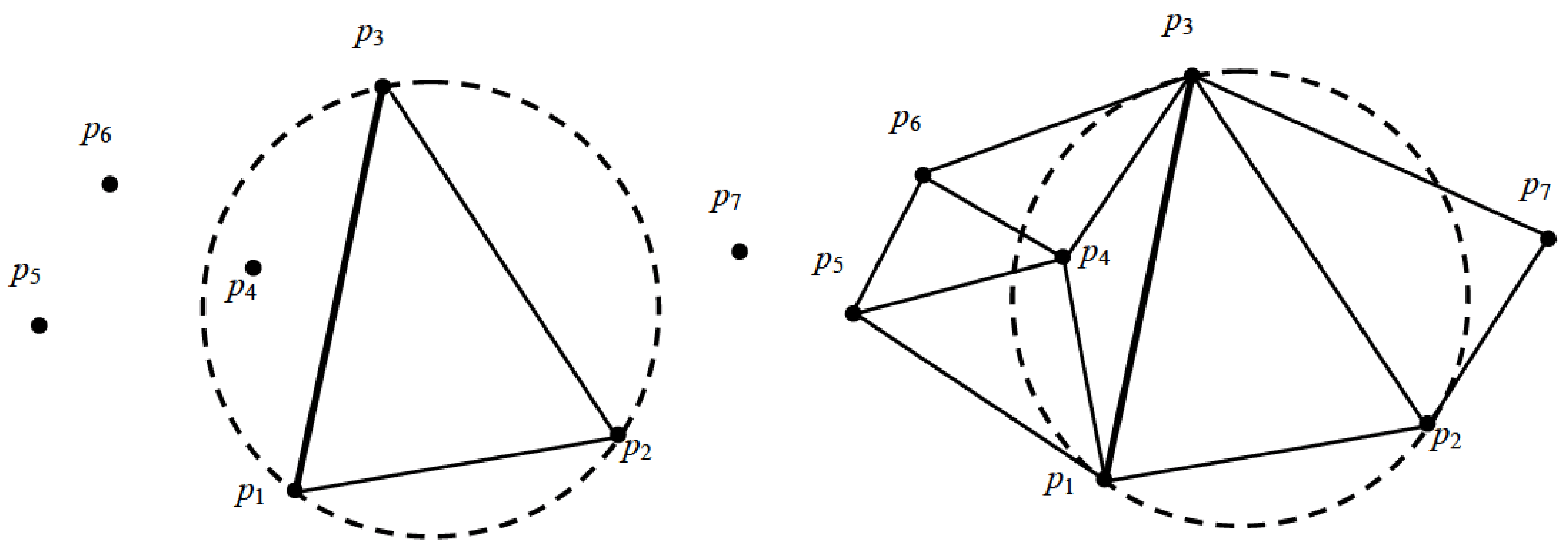


图 4.1　约束边产生遮挡的情况

图 4.2　图 4.1 中平面直线图的约束 Delaunay 三角剖分

定义 4.3　对给定的平面直线图 G=(P，E)，其约束 Delaunay 三角剖分是对 G 的约束三角剖分，但在每个三角形 t 的外接圆内，均不存在被 t 的三个顶点同时可见的 P 的其他点，即该三角剖分遵循约束空圆准则，我们称这样的三角剖分为约束 Delaunay 三角剖分（Constrained Delaunay Triangulation，CDT）。其中的三角形称为约束 Delaunay 三角形。约束 Delaunay 三角形中，与一条边相对的顶点称为该边的 DT 点。

为了叙述方便，我们用 CDT(G)表示 G=(P，E)的约束 Delaunay 三角剖分。CDT(G)覆盖 G 中所有约束边和离散点，即在 P 中的点都要成为 CDT(G)中三角形的顶点，E 中的边都要成为 CDT(G)中三角形的边（如图 4.2 所示）。由于约束边在 Delaunay 三角剖分过程中起到遮挡作用，则其 Delaunay 三角剖分有条件地继承了离散点集的 Delaunay 三角剖分中三角形外接圆是空圆的要求。

第 2 章中介绍了离散点集的 Delaunay 三角剖分具有最小角最大特性（等价于最大空圆特性），且其是离散点集的所有三角剖分中最优的三角剖分。CDT(G)也有条件地继承了这些特性。

性质 4.1　在 CDT(G)中，一个三角形 t 是约束 Delaunay 三角形当且仅当 t 的外接圆是约束空圆，即 t 内部均不存在被 t 的三个顶点都可见的 P 中的其他点。

性质 4.2　在 CDT(G)中，两个点 p_i、p_j 构成的边 p_ip_j 是 Delaunay 边当且仅当 p_i、p_j 互为可见，且存在一个过 p_i、p_j 的圆，该圆中不存在任何相对于 p_i、p_j 都可见的 P 中的其他点。

性质 4.1 和性质 4.2 可用于构造 CDT(G)。由于 CDT(G)属于点集 P 的一般三角剖分的一种方案，根据第 2 章中所介绍的离散点集的 Delaunay 三角剖分的性质，可得到下面的性质 4.3。

性质 4.3　CDT(G)最多有 $3n-6$ 条边，最多有 $2n-5$ 个 Delaunay 三角形。

在实际应用中，平面多边形（含有 1 条外边界和 $h\geqslant0$ 条内边界）中含有 $m\geqslant0$ 条约束边和 $k\geqslant0$ 个离散点的，是常见的一类平面直线图。其中，E 包含约束边和多边形域的边，P 包含离散点、约束边的顶点和多边形的顶点（如图 4.3 所示）。要在多边形域内建立该平面直线图的约束 Delaunay 三角剖分，即将多边形实现约束 Delaunay 三角剖分（如图 4.4 所示），其中所有约束 Delaunay 边都在平面多边形域的边界上或内部[Zeng2005a]。

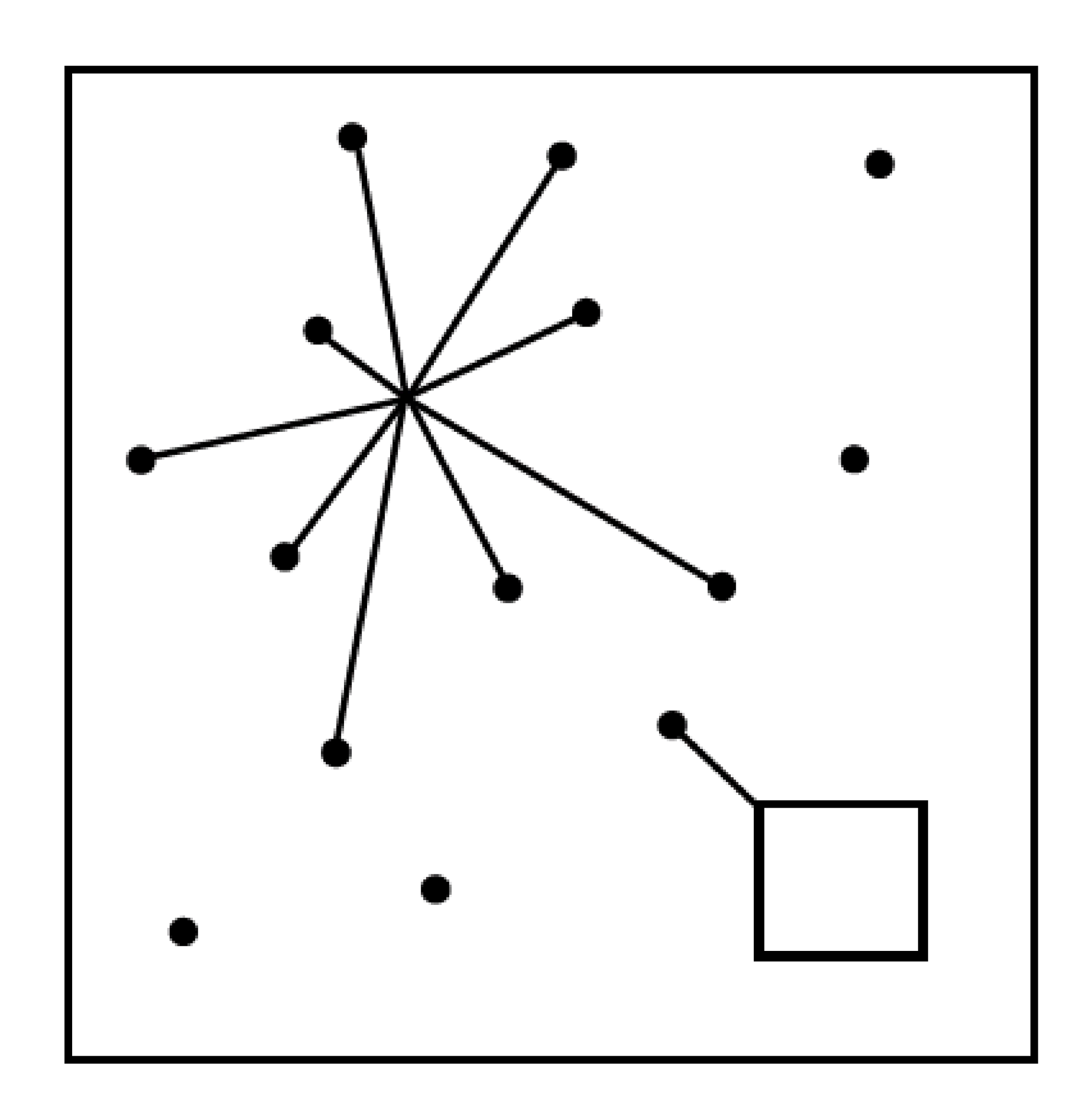


图 4.3　一个多边形域中的 PSLG

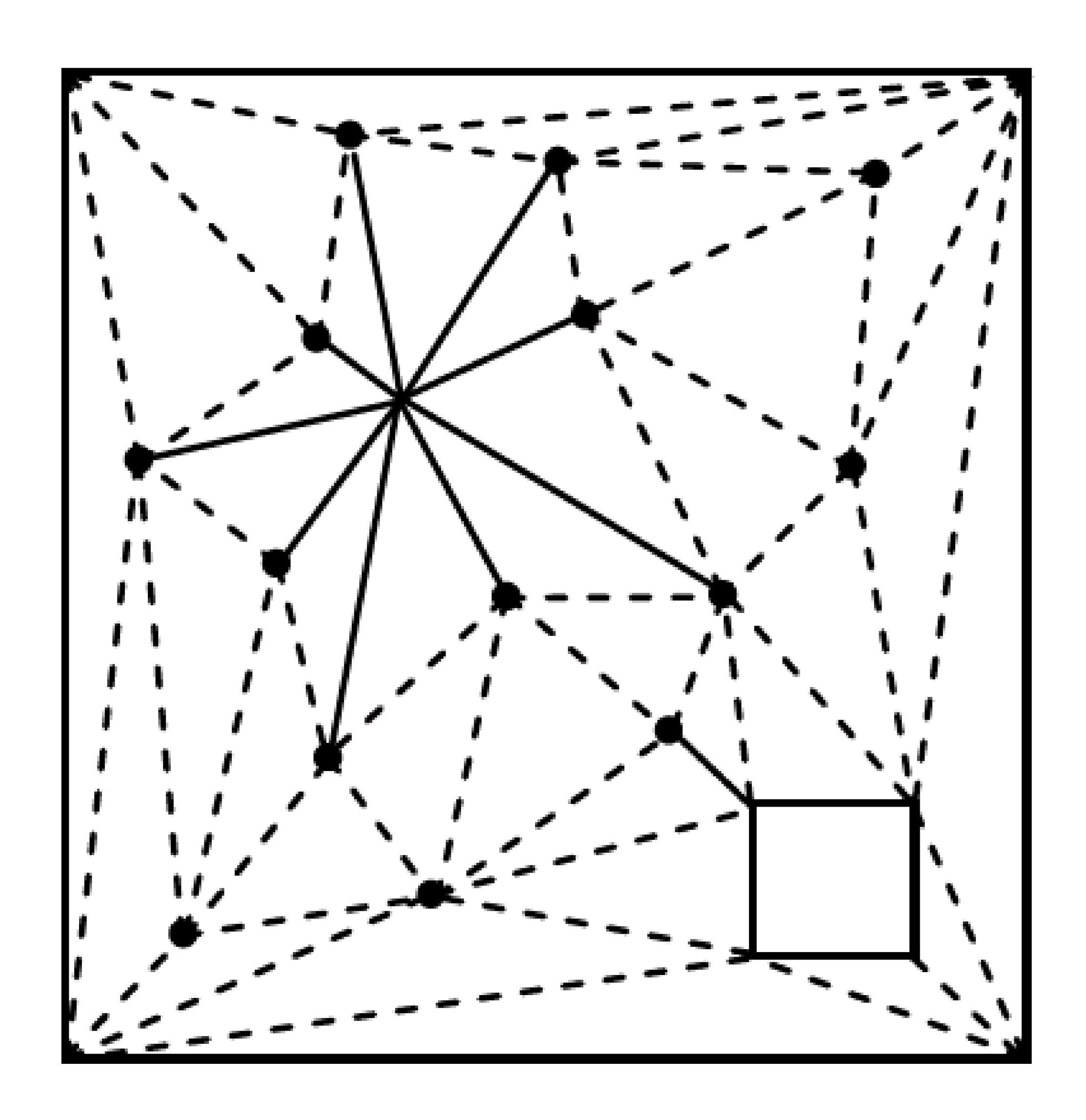


图 4.4　图 4.3 中的 PSLG 的 CDT

对于任意平面直线图，可以先计算其凸包，即可得到一个平面多边形域，该平面多边形域的约束 Delaunay 三角剖分即为该平面直线图的约束 Delaunay 三角剖分；也可以在其外面加一个矩形或三角形，转化成多边形域的情况。所以，本章后面主要针对多边形域的情况进行介绍。

4.2 构造方法

文献[WangJY2011]介绍了几种关于平面直线图和多边形的约束 Delaunay 三角剖分算法。本节介绍一种平面多边形域中平面直线图的快速约束 Delaunay 三角剖分算法。该算法也适用于一般平面直线图和一般多边形的约束 Delaunay 三角剖分[Zeng2005a]。算法不会生成区域外的三角形；对存在折线、离散点以及含“洞”的情况不需要特殊处理。实验表明，对于随机生成的多边形域的三角剖分速度快，平均计算时间呈近似线性。同时，算法能够有效地处理顶点、约束边的重叠情况和多点共线、四点共圆等退化情况，从而能够稳定、正确地处理各种域信息，并且其效率满足实际应用的需求，因此具有广泛的应用价值。

针对任意平面多边形域，算法采用增量思想，借助于均匀网格，在局部范围内快速生成约束 Delaunay 三角形：首先，用网格记录顶点和线段的空间位置和它们之间的大致邻接关系；然后，以平面多边形域的任意一条外边界边作为初始边，在附近区域搜索 DT 点以计算其约束 Delaunay 三角形，并用堆栈记录新生成且未被处理过的内部边；接下来，从堆栈中任取一条内部边，重复上面的过程，直至堆栈为空，从而实现整个平面多边形域的 CDT。

算法 4.1 基于均匀网格的平面多边形域的快速约束 Delaunay 三角剖分算法

（1）计算多边形域的包围盒。令 b_w 和 b_h 分别为多边形域包围盒的宽度和高度。

（2）建立单元大小为 $E \times E$ 的均匀网格，并基于此建立数据结构，将多边形域的顶点和边的信息放入其中。其中 $E=\frac{b_w \times b_h}{n}$，$n$ 为多边形域的顶点数。

（3）取任意一条外边界边 $p_i p_j$。

（4）计算 DT 点 p_k，构成约束 Delaunay 三角形$\triangle p_i p_j p_k$。

（5）如果新生成的边 $p_i p_k$ 不是约束边，则

（5.1）若其已经在堆栈中，则将其从中删除；

（5.2）否则，将其放入堆栈。

（6）若堆栈不空，则从中取出一条边，设为 $p_i p_j$，转步骤（4）；否则，算法停止。

下面主要介绍如何构建均匀网格并建立数据结构，以及如何计算一条边的 DT 点等。

1. 创建网格结构

算法首先建立单元大小为 $E \times E$ 的均匀网格，其中 $E=\sqrt{\dfrac{b_w \times b_h}{n}}$，$n$ 为多边形域的顶点数，b_w 和 b_h 分别为多边形域包围盒的宽度和高度，$b_w = x_{\max} - x_{\min}$，$b_h = y_{\max} - y_{\min}$。在网格结构中，为网格单元 b 分别建立一个 b 内的多边形顶点链表 $Point(b)$ 和一个经过 b 的多边形边链表 $Edge(b)$。整个网格结构使用二维数组来表示。若将一个顶点 $p(x，y)$ 放入相应的网格单元，其对应的数组下标为（int（$x-x_{\min}/E$），int（$y-y_{\min}/E$）），这里 $x_{\min}$，$y_{\min}$，$x_{\max}$，$y_{\max}$ 是点集中各点坐标的最小和最大 x、y 值；若将多边形域中的一条边界边放入其经过的网格单元，其对应的数组下标可通过光线跟踪技术快速计算得到。

整个平面多边形域使用一个顶点链表 *polygon* 来记录，约束边由顶点间的连接关系隐含表示，分别记录外边界和内边界、折线、离散点（三者顺序可调）；各边界和折线上的顶点按约定顺序排列。各个顶点的真实信息仅存在于 *polygon* 各结点中；网格单元的顶点链表结点和边链表结点都通过指针引用 *polygon* 结点获得顶点或相关联的边信息。这样既节省空间，又能够保证数据一致。

2. 计算 DT 点

将计算一条边的 DT 点的过程称为一趟。每趟包含两步：寻找初始可见点和确定 DT 点。该过程中借助均匀网格，将 DT 点的搜索限制在局部范围

内，加快计算速度。

（1）寻找初始可见点

假定当前需要处理的边为 $p_1 p_2$（如图 4.5 所示）。那么，以 $p_1 p_2$ 的中点 p_{mid} 所在的网格单元为中心，按照螺旋式扩散的顺序在有向边 p_1p_2 左侧搜索网格单元，对其顶点链表中各结点做相对 p_1p_2 的可见性判断，直至找到一个可见点 p_3，称其为初始可见点。每扫描一个网格单元，就将其标记为当前趟数。当网格单元中包含多个 p_1p_2 的可见点时，选取使得$\angle p_1pp_2$ 最大的那个可见点 p。

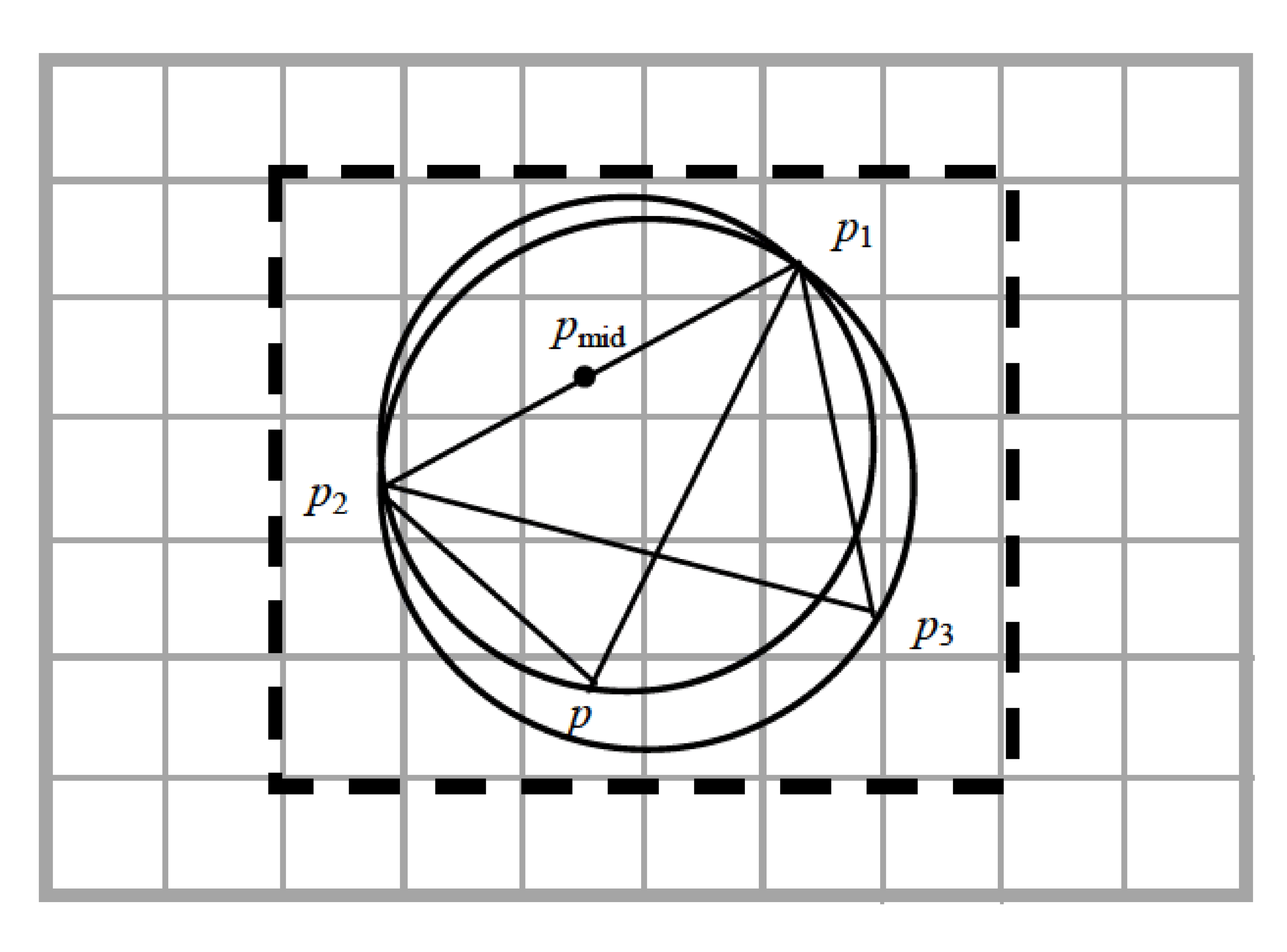


图 4.5　计算 DT 点

这里，称 p_3 为 p_1p_2 的可见点，必须满足下面三个条件：

① p_3 在有向边 p_1p_2 的左侧；

② p_3 与 p_1 可见，即对于 p_1p_3 所穿过的所有网格单元中记录的任意一条边 e，有 $e\cap p_1p_3=\varnothing$；

③ p_3 与 p_2 可见，即对于 p_2p_3 所穿过的所有网格单元中记录的任意一条边 e，有 $e\cap p_2p_3=\varnothing$。

（2）确定 DT 点

对于均匀网格上的任意一条有向边 p_1p_2，确定其 DT 点的过程如下：

① 构造△$p_1p_2p_3$的外接圆 C(p_1，p_2，p_3)及其网格包围盒 C_B(C(p_1，p_2，p_3))（如图 4.5 虚线所示）；

② 依次访问网格包围盒 C_B(C(p_1，p_2，p_3))内的每个网格单元：对没有当前趟数标记的网格单元进行搜索，并将其标记为当前趟数。若某个网格单元中存在可见点 p 并且$\angle p_1pp_2>\angle p_1p_3p_2$，则令 $p_3=p$，转步骤（1）；否则，转步骤（3）；

③ 若 C_B(C(p_1，p_2，p_3))内所有网格单元都已被标记为当前趟数，也即 C(p_1，p_2，p_3)内无可见点，则 p_3 为 p_1p_2 的 DT 点。

当 p_1p_2 恰有两个可见点 p_3、p_4 满足$\angle p_1p_3p_2=\angle p_1p_4p_2$，即四点共圆时，在共圆点 p_3 被选作 DT 点后，四点共圆不复存在；新生成的 Delaunay 三角形内部边 p_1p_3 或 p_3p_2 中，必有一条与 p_4 可见，对它必然选中 p_4 作为 DT 点，从而实现四点共圆下的 CDT。由此可见，共圆可见点中的一个一旦被选作 DT 点，其余的对此次搜索不再起作用；正是因为计算的不重复性以及增量式生长结构的前进性，避免了死循环，自动有效地解决了四点共圆的情况。多点共圆的情况与此类似。另外，通过动态更新包围盒，可以有效地收缩可见点查找范围；并充分利用网格单元搜索标记，避免重复操作以加快速度。

3. 堆栈结构的维护

这里，算法采用堆栈数据结构存放需要处理的边界边或内部边。每趟的 DT 点计算都是从堆栈中取出一条边开始进行，如果堆栈为空，则算法停止。算法在开始时，选择任意一条边界边来初始化堆栈；其后，每当生成一个 Delaunay 三角形时，判断新生成的内部边是否在堆栈中，若是，则从中删除，否则加入堆栈。

如图 4.6（a）所示，最初堆栈中只有边界边 p_1p_2；然后取出 p_1p_2 作为初始边，计算其 DT 点 p_3，生成一个新的 Delaunay 三角形，并将内部边 p_1p_3 放入堆栈中（见图 4.6（b））；取出 p_1p_3 为当前边，采用同样的方法计算出其 DT 点 p_4（见图 4.6（c））。每次生成的三角形中的新内部边（如图 4.6（c）中的 p_4p_3 和 p_1p_4）的方向必须和邻接的边界边方向保持一致，使得形成的新

边界的左侧始终为未处理区域。

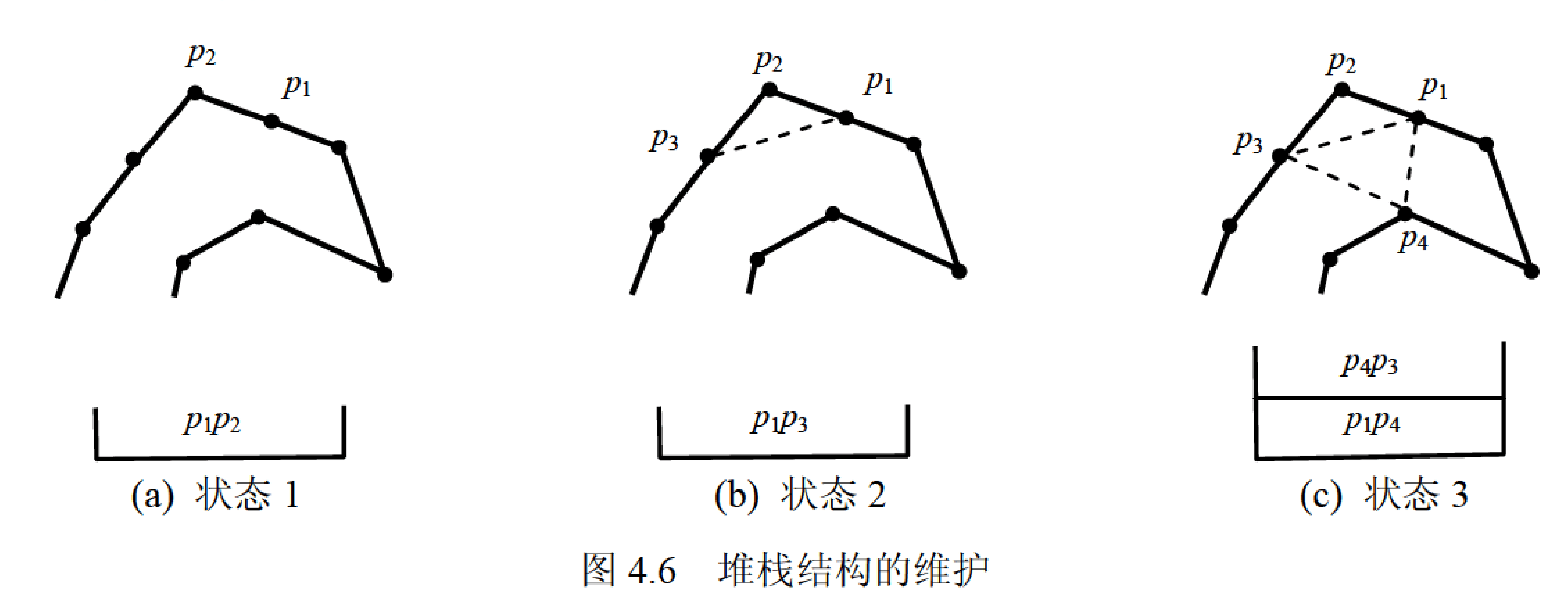


图 4.6 堆栈结构的维护

4.3 应 用 实 例

本节介绍基于平面直线图的约束三角剖分的几类应用，主要包括带状图像的骨架计算、在线手写体识别、点定位、可见性计算等。

4.3.1 带状图像的骨架计算

在 2.3.3 节，我们介绍了一种计算带状图像的算法。该算法首先计算图像边界顶点的 Delaunay 三角剖分，然后再用图像的边界对其进行裁剪，得到多边形的三角剖分，再进行骨架计算。该算法得到的三角剖分的部分顶点可能不是原来多边形的顶点，而且该三角剖分不是严格的约束 Delaunay 三角剖分。

针对文字、工程图、指纹等带状图像边界形成的带状多边形，根据其特点，本节对 4.2 节介绍的多边形域约束 Delaunay 三角剖分算法进行了改进，并以此为基础设计实现了一种有效的带状图像骨架化算法[Zeng2004]。算法除了 4.2 节中所述的对多边形域的 CDT 的优势之外，针对这一类特定的带状多边形，算法中每个网格单元只存储顶点链表信息，存储空间大量减少，易于维护；算法更易于理解和实现；对于多边界多边形与简单多边形能统一处理，不需要额外操作。计算出的骨架要优于 2.3.3 节的算法，但需要计算均匀网格并设计额外的数据结构。

给定一幅带状图像 I_b，其近似笔划宽度为常量 D；假定常量 E 正比于 D，用于 I_b 的连续边界矢量化，使得每条边长不大于 E。本节中，将这样得到的多边形称为带状多边形。对于复杂多样的带状图像，它们的带状多边形通常为多边界多边形。带状多边形的内、外边界分别按照顺、逆时针表示，以至于多边形域的内部总在边界左侧。

算法 4.2　带状图像骨架化算法

（1）建立单元大小为$E \times E$的均匀网格，基于此建立数据结构，并放入多边形域的顶点信息。

（2）选择多边形的任意一条外边界边作为初始边，压入堆栈。

（3）从堆栈弹出一条边，在该边附近的网格单元内搜索其DT点，生成Delaunay三角形，并将该三角形中新生成的内部边压栈。

（4）重复第（3）步，直至栈空，从而实现带状多边形的Delaunay三角剖分。

（5）根据每个三角形的类型计算其对应的骨架线，并将它们连接生成整个图像的骨架。

在算法4.2中，第（5）步采用2.3.3节中的方法，根据每个三角形的类型计算其对应的骨架线，并将它们连接生成整个图像的骨架。下面主要介绍如何构建均匀网格并建立数据结构，以及如何计算一条边的DT点等。

1. 创建网格结构

网格结构的具体构造方式见 4.2 节中的算法 4.1。这里讨论如何基于网格建立合适的数据结构。

在算法 4.1 中，各网格单元通常包含一个经过该网格单元的多边形边链表和一个所有位于该网格单元内的多边形顶点链表。这里，由带状多边形边长的均等性以及网格宽度与边长的约束关系可知，与一条线段相交的边界边可以在局部范围内根据边界边的顶点找到。

对均匀网格中的任意线段 p_ip_j，令 $B_m(p_ip_j)$表示其矩形包围盒。所有与 $B_m(p_ip_j)$相交的网格单元组成 p_ip_j 的网格包围盒，表示为 $C_B(p_ip_j)$。它所覆盖的区域及其八连通区域构成 p_ip_j 的扩展包围盒，表示为 $E_B(p_ip_j)$，如图 4.7 所示。可以证明，在单元大小为 $E\times E$ 的均匀网格中，多边形 P 的每条边长不大于 E，那么对其任意两个顶点 p_i 和 p_j，所有与 p_ip_j 相交的多边形边界边都在 p_ip_j 的扩展网格包围盒内。因此，每个网格单元只需要保存多边形顶点信息，与其相连接的边界边很容易找到。所有的多边形由一个链表表示，存储每个顶点的信息。

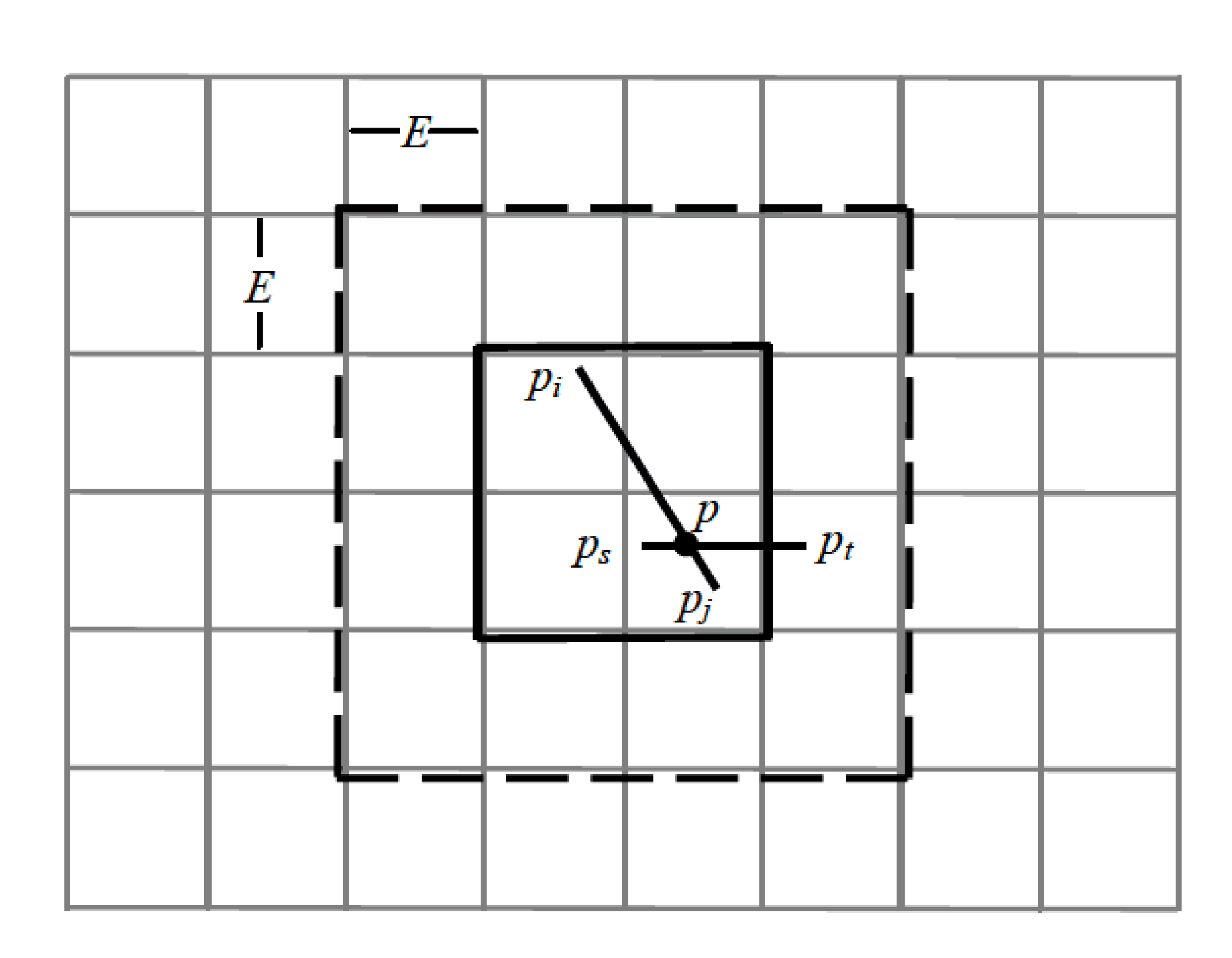


图 4.7　网格包围盒 $C_B(p_ip_j)$（实线矩形域）及扩展包围盒 $E_B(p_ip_j)$（虚线矩形域）

2. 计算 DT 点

同样地，计算一条边的DT点包含两步：寻找初始可见点和确定DT点。由于带状多边形的结构特性以及网格结构的特殊构造，点关于边的可见性计算大大减少。可见性判断与4.2节中的方法有所不同，其他内容基本一致。

若点 p_3 是边 p_1p_2 的可见点，所必须满足的条件相应地变化为：

（1）p_3 在边 p_1p_2 的左侧；

（2）$\bigcup\limits_{\forall c\subset E_B(p_1p_3)}\bigcup\limits_{\forall p\in Vertex(c)}[p_{pre}p\cap(p_1p_3)]=\varnothing$，即 p_1 与 p_3 彼此可见；

（3）$\bigcup\limits_{\forall c\subset E_B(p_3p_2)}\bigcup\limits_{\forall p\in Vertex(c)}[p_{pre}p\cap(p_3p_2)]=\varnothing$，即 p_3 与 p_2 彼此可见。

条件（2）中，扩展网格包围盒 $E_B(p_1p_3)$覆盖所有可能与 p_1p_3 的开线段部分相交的边，且 c 为 $E_B(p_1p_3)$内的任意网格单元；$Vertex(c)$是在 c 内的顶点集合，p 是 c 内的任意顶点；p_{pre} 是 p 所属的多边形上的前一顶点。对每个顶点 p，仅需要用来索引同一方向的边即可，避免重复计算。这里选择前向边 $p_{pre}p$。条件（3）中情况类似。

3. 参数设置

对于一个确定的带状多边形，要求其每条边的长度不大于 E。算法 4.2 中设定的网格宽度等于给定常量 E。这样可以保证，与某边相交的所有边界边均在其扩展网格包围盒内，每个网格单元只需要记录多边形的顶点信息。另外，对于一个确定的带状图像，E 与 D 之间的比例关系很大程度影响了网格上的点分布，从而进一步影响带状多边形三角剖分的计算时间。考虑到对一个确定的带状图像和它的带状多边形，其笔划宽度 D 是一个固有的常量，可通过 D 设置 E 使得算法消耗时间减少。大量的实验结果表明，对于指纹、手写体文字以及大部分工业图案，一般 E 在 $2D$ 的附近取值时 Delaunay 三角剖分时间消耗最少。

4.3.2　在线手写体识别

在线手写输入作为一种良好的人机交互手段，其应用越来越普遍，因此在线手写体识别算法也得到广泛研究。鉴于不同人有着不同的书写习惯，同一字符会有成千上万种写法。当样本数量足够大时，通过统计的方法可以获取一些特性及规律，这往往蕴涵在使用的统计模型中。当模型训练及识别算法已确定、模型参数以及训练样本固定时，不同的特征提取方法很大程度上决定了识别效率，在整个识别流程中起着关键作用。

Delaunay 三角剖分作为一种得到广泛应用的区域剖分手段，有其本身固有的优势：最小角最大、空外接圆等全局性，以及插入、删除顶点只引起局部修改等鲁棒特性。在排除退化的情况下，Delaunay 三角剖分网格唯一，可保证计算的稳定性；另外，三角形本身是在 Delaunay 标准下生成的，受周围乃至全局顶点和约束边的影响，代表几何和拓扑结构，具有很强的特征标识

性。本节介绍一种基于 CDT 的在线手写字符的特征提取方法[Zeng2006，Zeng2005b]，其用 CDT 方法将手写体采样点的周围区域有规律地进行剖分，然后描述采样点的全局特征。

下面首先介绍在线手写字符，然后分别介绍两种不同的 CDT 方法以及 Delaunay 三角形特征描述器，最后给出实验结果与分析。

1. 在线手写字符

在线手写字符是在统一的硬件采样方式下生成的，所有硬件采样点之间仅存在一维的前后邻接关系，可通过采样点的坐标信息重构出在线手写字符图形，每相邻两点间的实笔划段均可看作约束边。

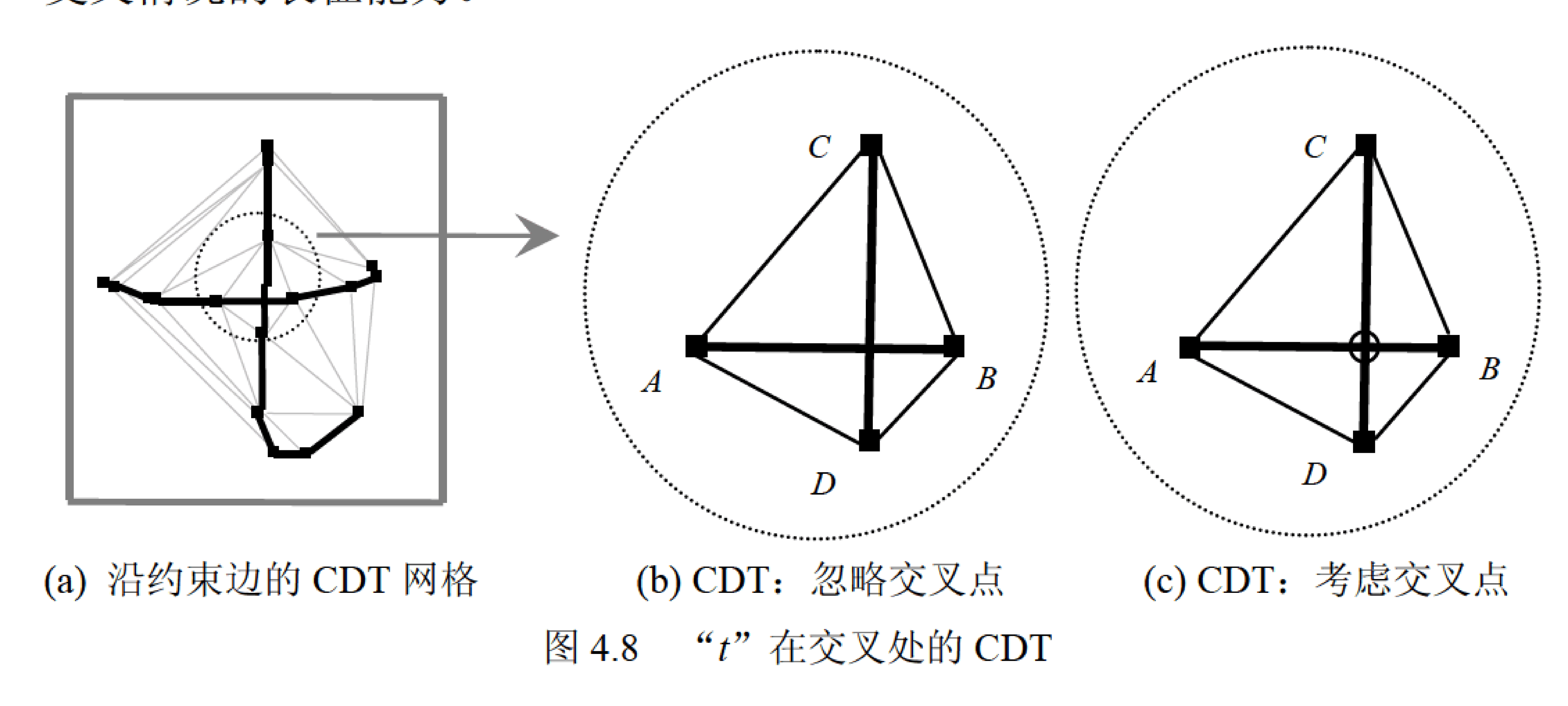
对在线手写字符提取特征前，由于书写风格的千差万别，导致了采样时的时域空间和空域空间不一致以及在线手写图形的复杂多样性，因此硬件记录的原始数据必须经过若干预处理步骤。常规预处理一般包括尺寸大小调整、倾斜旋转归一化、平滑、插值、重采样等。尽管硬件仅记录下笔落下的线段，但是虚拟笔划及笔划的起始点信息仍可考虑用来丰富特征信息；还可通过提取控制点来获取更加有效的特征，减少不必要的冗余信息，降低存储，提高速度。

所谓控制点，即指能够基本标识图形局部几何特征的顶点；控制点序列某种程度上代表着整个图形的拓扑结构。它的计算方法并不十分严格，一般取：（1）曲率变化大的顶点，（2）笔划起始点，在（3）夹角累计和阈值以及（4）点距阈值约束下的顶点等。不同的限制条件下产生的采样点位置及数目都不同。显然，采样点越密集，涵盖的信息也就越丰富；但存储量大，耗时也较多。在条件（1）、条件（2）下生成的控制点仅在极值处采样，称之为极值控制点；在条件（1）～（4）下生成的控制点，点数较多，较均匀，称之为均匀控制点。均匀控制点是在极值控制点和原始采样点之间的折中处理。

另外，由于在线手写图形具有极为复杂的多样性以及字符本身的形体构造特点，很难避免采样点的重叠和笔划线段的交叉、重叠等状况的发生。

对交叉点的处理，一般有两种处理方式：考虑交叉点的存在和忽略交叉点的存在。目前众多的在线手写体识别系统一般不对交叉点做任何特殊处理，即不考虑笔划线段是否交叉，忽略交叉拓扑信息。相应的特征提取方式一般计算简单快捷，但也丢失了一定程度上的结构特征。若考虑交叉点的存在，需要引入浮点数运算来判断交叉情况，虽然考虑到了手写体的图形结构特性，但会增加计算的复杂性，降低实现的鲁棒性。

在使用 Delaunay 三角形来获得一条约束边的更全局的特征之前，应尽量有效地计算与组织在线手写字符图形的 Delaunay 三角形。鉴于此，本算法给出两种 CDT 方法——PSLG 的 CDT 和沿约束边的 CDT（分属于上述两种交叉点的处理方式，如图 4.8 所示）——来探讨 Delaunay 三角形特征描述器对交叉情况的表征能力。



(a) 沿约束边的 CDT 网格　　(b) CDT：忽略交叉点　　(c) CDT：考虑交叉点

图 4.8　“t”在交叉处的 CDT

2. PSLG 的 CDT

在考虑交叉点的存在时，首先要计算交叉点，将交叉点插入并分割原字符图形，形成 PSLG，这个过程称为 PSLG 转换；然后应用 4.2 节中的 PSLG 的 CDT 算法。

由于前述 CDT 算法适用于任意封闭或开放的 PSLG 形式的图形，而在线手写体一般是非 PSLG 图形（比如存在笔划线段交叉情况，如图 4.8（a）所示），其本身通常不能构成严格意义上的区域，因此，虽然对算法稍作调

整（否则有可能出现死循环）可以处理这种情况，但结果中仍会存在重叠的三角形。所以，需要对输入信息进行 PSLG 转换，以便直接利用前面的区域 CDT 算法。如图 4.8（b）中，线段 *AB* 与 *CD* 相交，在忽略交叉点存在时，仅对 *AB* 和 *CD* 做计算，分别生成△*ABC* 和△*ABD* 及△*CDB* 和△*CDA*，三角形彼此重叠，在交叉处不构成严格意义上的三角剖分；若计算交叉点 *O*，如图 4.8（c）所示，线段 *AB* 与 *CD* 被分割，线段 *AO*、*OB*、*CO* 和 *OD* 分别被计算，生成的三角形彼此邻接，覆盖交叉区域。这种方法引入了笔划线段的交叉点信息，可获得更丰富的特征。

对于一个确定的在线手写体图形，这里不妨用 *G* 表示，其线段是按照时序记录的，有着固定的顺序；假设 *G* 含有 *n* 条线段(也即约束边)，其线段集合表示为 $E(G) = \{e_1，e_2，…，e_{n-1}，e_n\}$，那么从 e_2 开始，依次对 $e_i(1< i \leqslant n)$ 做如下操作：分别与其前面所有线段 $e_j(0 <j< i)$，判断是否交叉或重叠，并经交叉或重叠转换保证两条线段形成的图形序列化。整个 PSLG 转换需要 $O(n^2)$ 的时间复杂度。类似于前面 CDT 算法中使用的均匀网格，也可通过网格单元边链表以及当前线段所通过的网格单元集合，来寻找与当前线段可能交叉或重叠的前面的线段，减少计算次数，节省时间。算法先求取控制点，然后在其重构后的图形上进行 PSLG 转换，这样使得图形边数以及判断次数明显减少，并且同时直接记录下交叉点信息。

笔划线段交叉大致分四种情况：（1）完全交叉（如图 4.9（a）所示）；（2）一条线段的端点落在另一条线段上（如图 4.9（b）所示）；（3）两条线段的两个端点重合（如图 4.9（c）所示）；（4）一条线段的端点距另一条线段相当近（小于阈值 ε）（如图 4.9（d）、图 4.9（e）所示）。相应解决办法为：情况（1）将交叉点插入两线段中，这里所谓点的插入即是指每条线段被分割为两条邻接的线段；情况（2）将落定在线段上的端点插入该线段中；情况（3）中两条线段拥有同一个端点，相互邻接，不做其他转换；情况（4）计算两条线段的实际交叉点，并以此取代四个端点中最靠近它的那个端点，同时将它插入另一线段中。

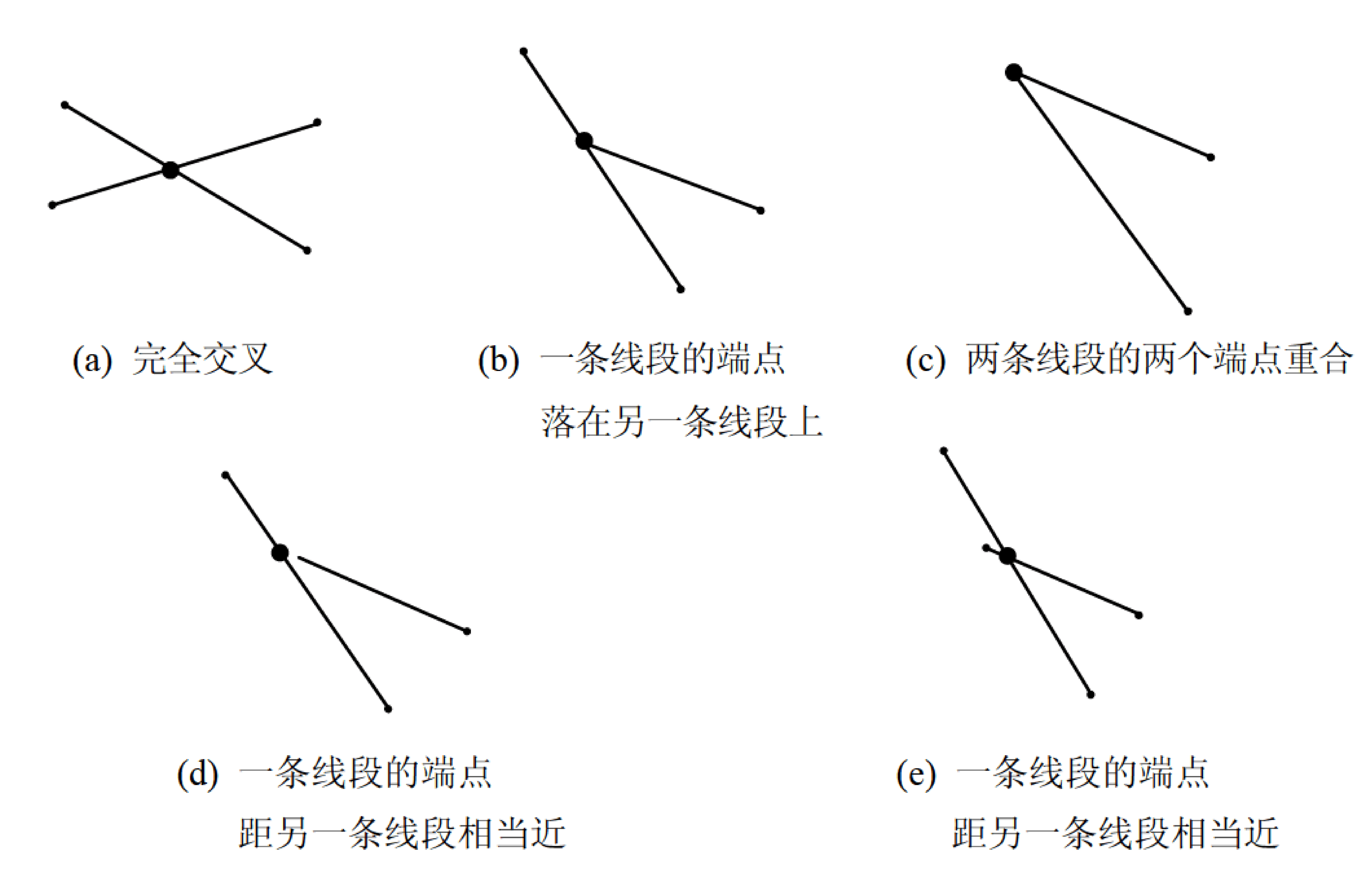


图 4.9　线段交叉情况，较大的黑色圆点表示实际交叉点

笔划线段重叠大致分四种情况：（1）完全重叠；（2）部分重叠；（3）完全包含；（4）部分包含，一端重合。分别如图 4.10（a）～图 4.10（d）所示。解决办法是通过端点排序，将公共部分的端点互相插入，使得原始线段的公共部分在分割后保持端点一致。图 4.11 给出了一些手写字符图形形成的 PSLG 的 CDT 结果。

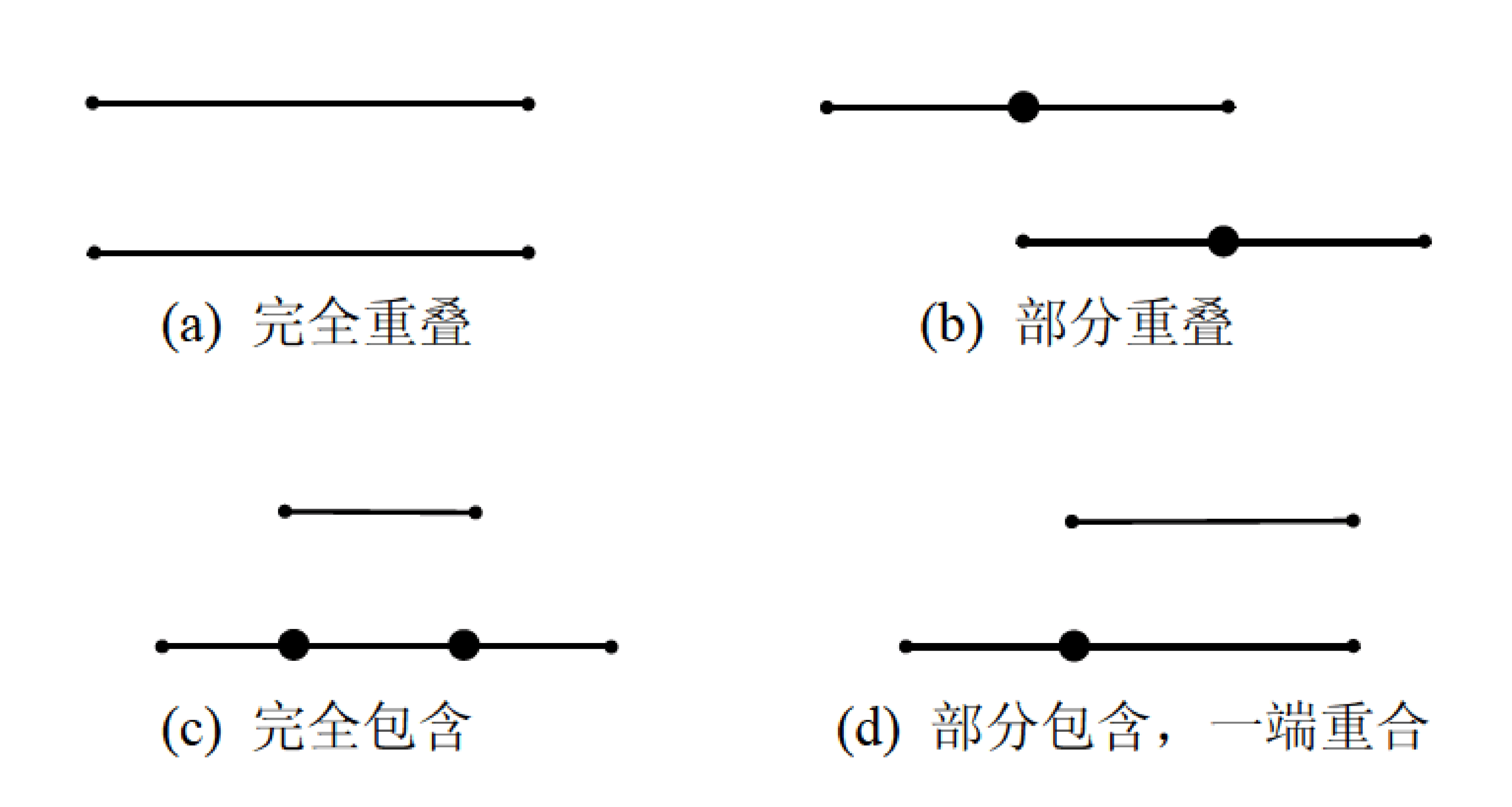


图 4.10　线段重叠情况，较大的黑色圆点表示相应线段的插入点

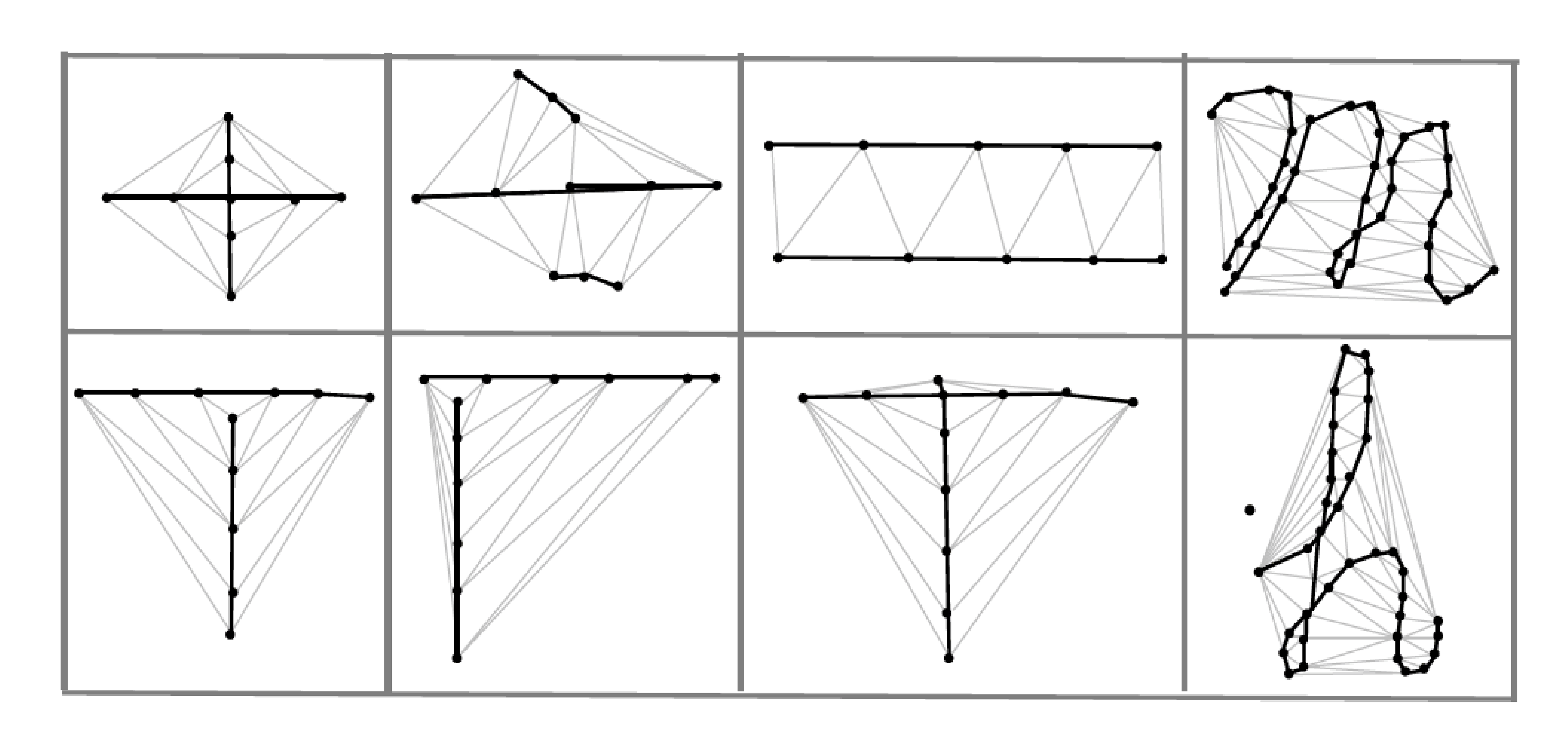


图 4.11　“+”、“=”、“*m*”、“*h*”等手写字符图形的 CDT

3. 沿约束边的 CDT

我们注意到，原始数据点均为整数类型。上述方法中，笔划线段的求交不可避免地引入浮点运算，导致交叉点的计算不够稳定。正是由于 PSLG 转换过程中的浮点运算引入的误差和不确定性，对调整得到的 PSLG 进行 Delaunay 三角剖分的过程中，对同一处几何元素，交叉、共线、共圆的判断结果往往出现误差。总体来说，在对在线手写字符图形进行 PSLG 转换及 CDT 时，要保证其计算的鲁棒性是相当困难的。

为了避免 PSLG 转换以增强鲁棒性，又考虑到仅对约束边所属的三角形进行特征提取，因此提出忽略交叉点存在的、沿约束边的 CDT 方法（可称为时序 CDT 方法）。具体地讲，省去交叉点的计算，直接沿笔划轨迹依次对各笔划线段利用 4.2 节所述的 CDT 方法，快速在其两侧生成 Delaunay 三角形。每条约束边最多对应两个 Delaunay 三角形（分属约束边两侧）。

不考虑约束边之间的位置关系，也即无须处理笔划线段交叉或重叠的情况。在对原始整数类型数据集上，三角剖分过程中碰到的共圆等退化情况的处理，均可通过加标记或数据“扰动”处理。这样，不会引入浮点误差，计算容易，且增强鲁棒性。此外，这种方法避免了不必要的三角形计算，与后面的 Delaunay 三角形特征提取阶段只对约束边提取特征的方式紧密结合。同

时，省略了 PSLG 转换中复杂的计算；虽然没有显式地记录交叉信息，降低了对交叉点的敏感度，但是由 Delaunay 三角形表示的上下文描述器拥有强烈的几何拓扑信息，所覆盖的全局性特征蕴含着一定程度的交叉信息。与第一种方法相比，沿约束边的 CDT 需要计算的 Delaunay 三角形数目减少。

4. Delaunay 三角形特征描述器

本节中，将三角形作为一种图形描述器，充分利用在线手写字符图形的几何信息及拓扑结构，既包含局部特征，又隐含着全局信息，这是因为 Delaunay 三角形的生成受全局顶点和约束边的影响。

具体来说，在前面 CDT 的基础上，沿约束边提取 Delaunay 三角形特征，并组合形成整个图形的特征序列。对一条约束边，考虑其本身及两侧 Delaunay 三角形信息，按某种特定的顺序组成该约束边的特征向量。各约束边的特征向量沿笔划轨迹的顺序构成特征向量序列，经矢量量化[Peter2002]转化为特征观察符号序列，以便进行隐 Markov 模型的训练和识别。

具体地，所构造约束边的特征向量共有 13 维，包括以下三部分。

（1）当前边：[边类型，边长 1，方向]，其中，边类型是根据当前边的邻接关系来对约束边进行分类来决定的，蕴含了 Pen-up/down 信息；

（2）左侧 Delaunay 三角形：[三角形类型，面积的开方，边长 2，边长 3，对角余弦]；

（3）右侧 Delaunay 三角形：[三角形类型，面积的开方，边长 2，边长 3，对角余弦]。

其中，三角形类型是根据三角形所包含的约束边的数目来决定的。CDT 三角网格中，交叉点处的三角形往往含有两条约束边，在拐角处也可能如此，因此，三角形类型一定程度上表征了笔迹交叉、笔锋急转等结构特征。

如图 4.12 所示，当前边为 AB，其斜角为 θ_{AB}，类型为 t_{AB}；AB 左右两侧的 Delaunay 三角形分别为$\triangle ABC$ 和$\triangle ABD$，其类型分别为 $t_{\triangle ABC}$ 和 $t_{\triangle ABD}$。那么，AB 的特征向量为：[t_{AB}，$|AB|$，θ_{AB}，$t_{\triangle ABC}$，sqrt($S_{\triangle ABC}$)，$|BC|$，$|CA|$，cos($\angle ACB$)，$t_{\triangle ABD}$，sqrt($S_{\triangle ABD}$)，$|BD|$，$|DA|$，cos($\angle ADB$)]。

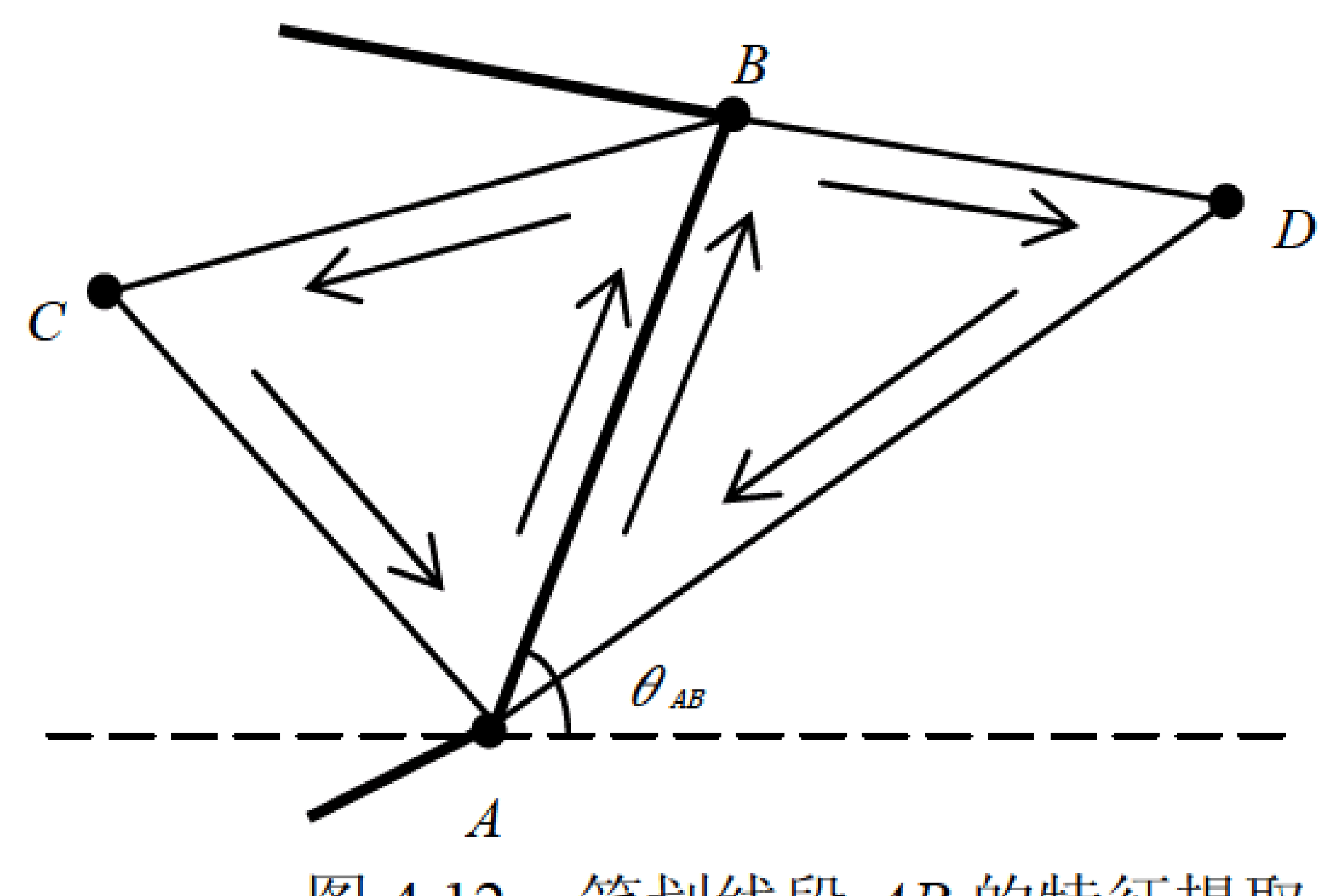


图 4.12　笔划线段 *AB* 的特征提取

5. 实验结果及对比分析

本节所使用的英文字符库由数学公式中所有数学函数单词中所涉及到的常用字符组成。表 4.1 列举了 10 类常用数学函数及 16 类常用字符，充分考虑了用户的语种、年龄、个人习惯等因素对书写方式的影响，由若干组用户在硬件手写设备上随机采样，得到 16 类、共 60 756 个字符训练样本和 5073 个测试样本。下面分别给出两种 CDT 方法下的实验结果及分析。

表 4.1　数学公式中的常用单词和字符

单词	arc	cos	csc	ctg	sec	sin	lim	ln	log	tg						
字符	a	c	d	e	g	h	i	l	m	n	o	p	r	s	t	x

（1）PSLG 的 CDT 识别结果

系统中使用 16 类字符样本中有两类因 PSLG 转换的结果不够鲁棒导致 CDT 不稳定而无法提取特征，不能参与训练。针对余下的 14 类字符样本，首先做常规预处理和 PSLG 转换，然后分别求取极值控制点和均匀控制点进行特征提取，来观察系统在训练和测试样本库下的总识别准确率。表 4.2 和表 4.3 列出了两种情况下的前 6 候选的识别准确率。可以看出，经过 PSLG 转换后，对均匀控制点提取特征产生的识别率要优于极值控制点的情况，第一候选识别率达到 91.15%。

表 4.2 经 PSLG 转换、极值控制点采样的 14 类字符字典下的识别率

Top	1	2	3	4	5	6	类数
总识别率	0.7920	0.8927	0.9324	0.9519	0.9638	0.9710	14

表 4.3 经 PSLG 转换、均匀控制点采样的 14 类字符字典下的识别率

Top	1	2	3	4	5	6	类数
总识别率	0.9115	0.9743	0.9899	0.9944	0.9965	0.9976	14

（2）沿约束边的 CDT 识别结果

针对系统中的 16 类字符训练样本，首先做常规的预处理及均匀控制点采样，然后采用沿约束边的 CDT 方法获得 Delaunay 三角形并进行三角形特征提取。如图 4.13 中所示的实例，表 4.4 列出了总的前 6 候选识别准确率。可以看出，相比经过 PSLG 转换的区域 CDT 思想，系统无需额外转换操作，使得鲁棒性增强，字典增加两类；同时减少了计算，加快了前期操作获得特征以进行识别，第一候选识别率也提高到了 92.06%。

表 4.4 经直接约束边 CDT、均匀控制点采样的 16 类字符字典下的识别率

Top	1	2	3	4	5	6	类数
总识别率	0.9206	0.9746	0. 9793	0.9819	0.9833	0.9841	16

表 4.5 不同特征组合下的 16 类字符集的识别率

No.	特征组合	Top1 总识别率
E1	Δx，Δy，$\Delta\theta$	0.8691
E2	E1 + Pen-up/down	0.8803
E3	E2 + Virtual Stroke	0.9004
E4	E3 + Sgn(x-max(x))[33]	0.8481
E5	E4 + θ ， as in [33]	0.8583
E6	θ，Pen-up/down，Curvature，Aspect，Curliness Linearity，Slope [Jaeger2000]	0.8494
E7	Delaunay 三角形特征	0.9206

（3）特征比较

针对系统中的 16 类字符训练样本以及同样的模型训练参数、分类器参数，不同的特征提取方法得到的系统识别率也不相同。系统中采用同样的常规预处理及均匀控制点采样，并基于离散隐 Markov 理论进行模型训练及设计分类器。表 4.5 列出了不同特征组合方式下的识别率。可以看出，特征 E7 要优于其他系统中传统的特征组合 E1～E6。

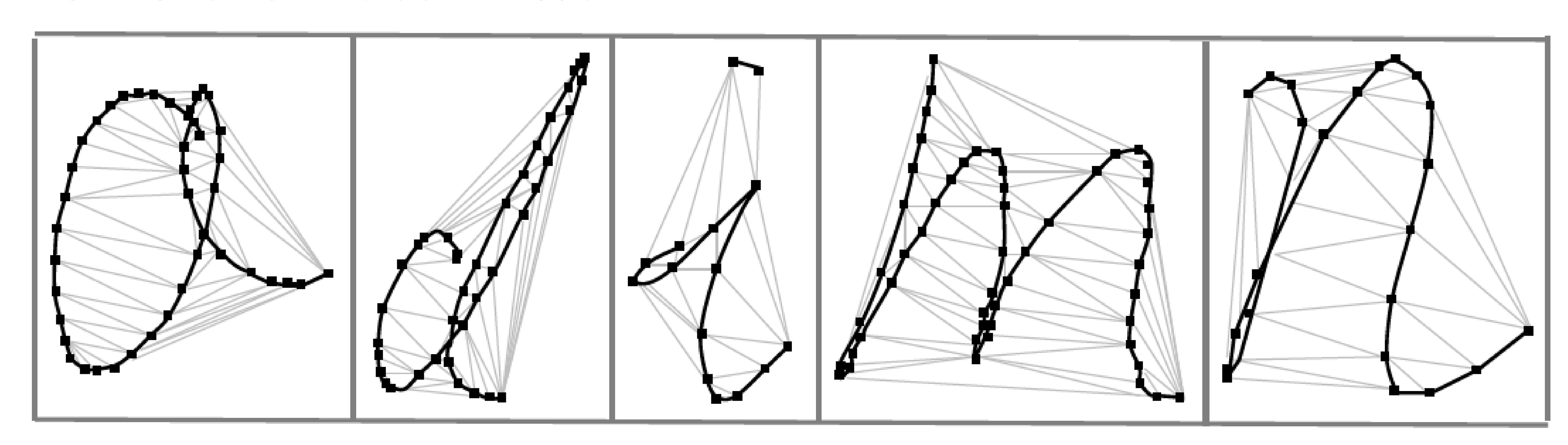


图 4.13　实例：在线手写字符图形的沿约束边的 CDT

4.3.3　点定位

空间定位查询是很多应用算法的基础，可以加快应用算法的速度。目前存在很多空间定位查询算法[WangJY2011]。本节主要介绍基于平面直线图的约束三角剖分的空间定位算法，它是由 D.G. Kirkpatrick[Kirkpatrick1983]1983 年提出的。它只用 $O(n)$ 空间存放预处理的结果，用 $O(\log n)$ 时间回答给定点是在哪一个区域内的询问。它完成预处理的时间是 $O(n\log n)$ 。

算法思想：首先计算 G 的凸包，如果该凸包不是一个三角形，可用一个足够大的三角形包围它，这样构造了一个外边界是三角形的平面多边形域。然后，对平面直线图 G=(P，E)进行约束三角剖分。定义图 G 的独立集是 P 的一个子集，要求子集中每一对结点在图 G 中均不是相邻的。如图 4.14（a）～图 4.14（d）中的黑点便是相应图的内点的独立集，且就内点而言它们是最大的独立集，也就是相应图的内点不存在包含更多点的独立集。最后，应用算法 4.3 构造点定位查询的数据结构。

算法 4.3　基于 CDT 构造点定位查询数据结构

输入：平面直线图 G=(P，E)的 CDT。F 是 CDT(G)中的三角形集合，$|F|$

是三角形个数。

输出：用于查询点定位的数据结构。

（1）*G* 的每一个三角形作为数据结构的一个结点；

（2）**while** (| **F** |⩾9){

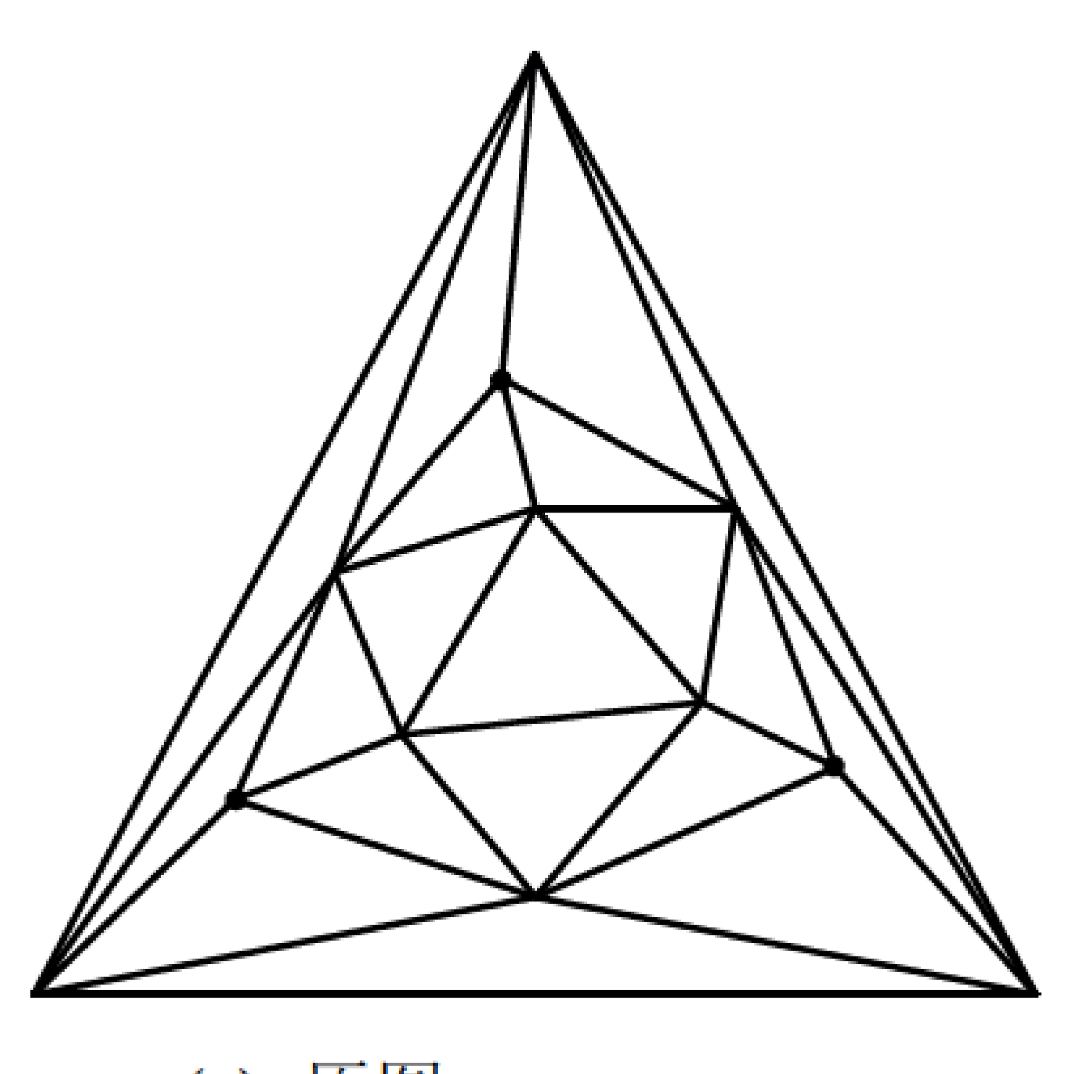
（3）从 *P* 的内点中找一个尽量大的独立集，且独立集中结点的度均小于9 ；

（4）删除独立集的结点及以这些结点为一个端点的边以及相应的三角形；

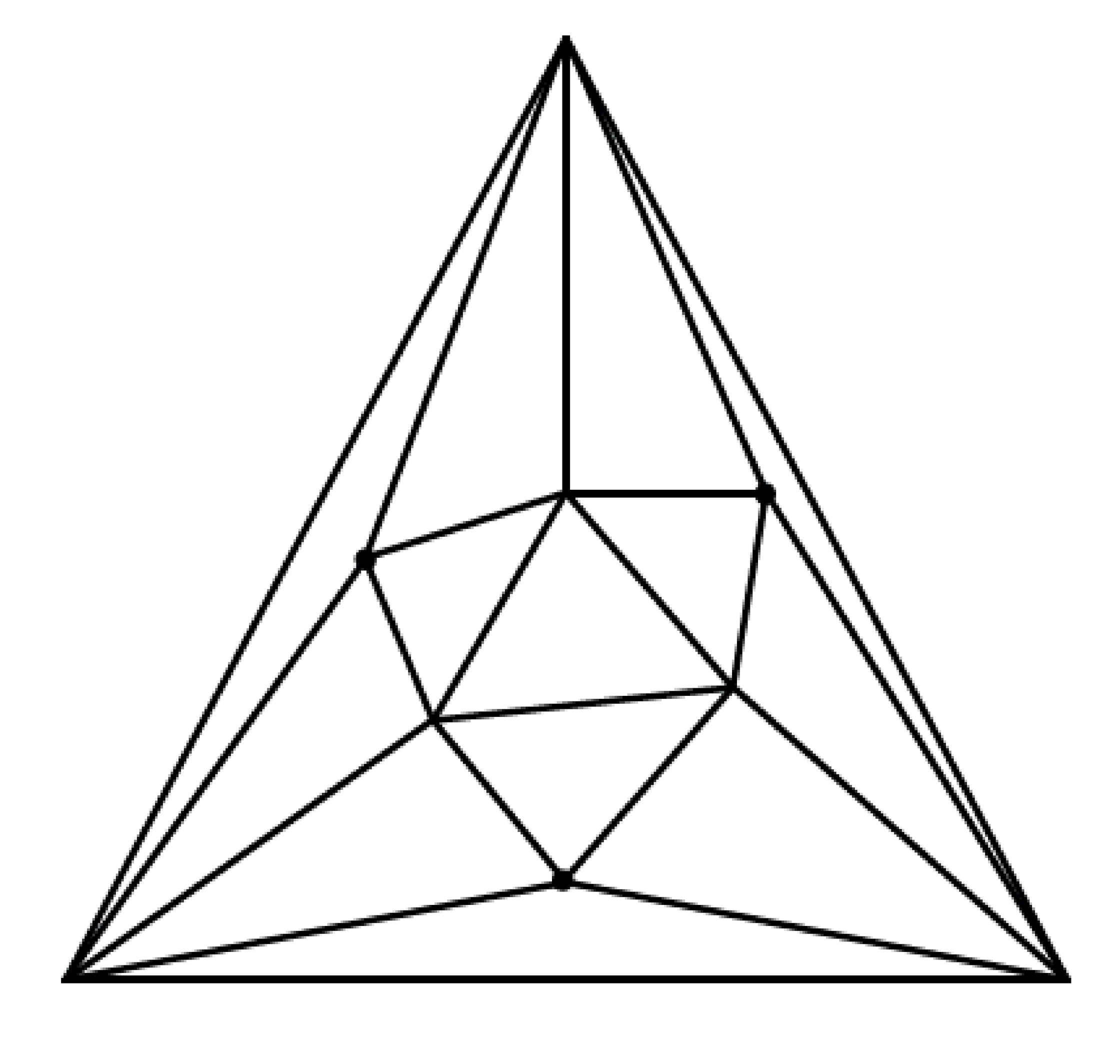
（5）在删除了结点和边后形成的多边形中，用对角线作三角形剖分；

（6）更新数据结构。

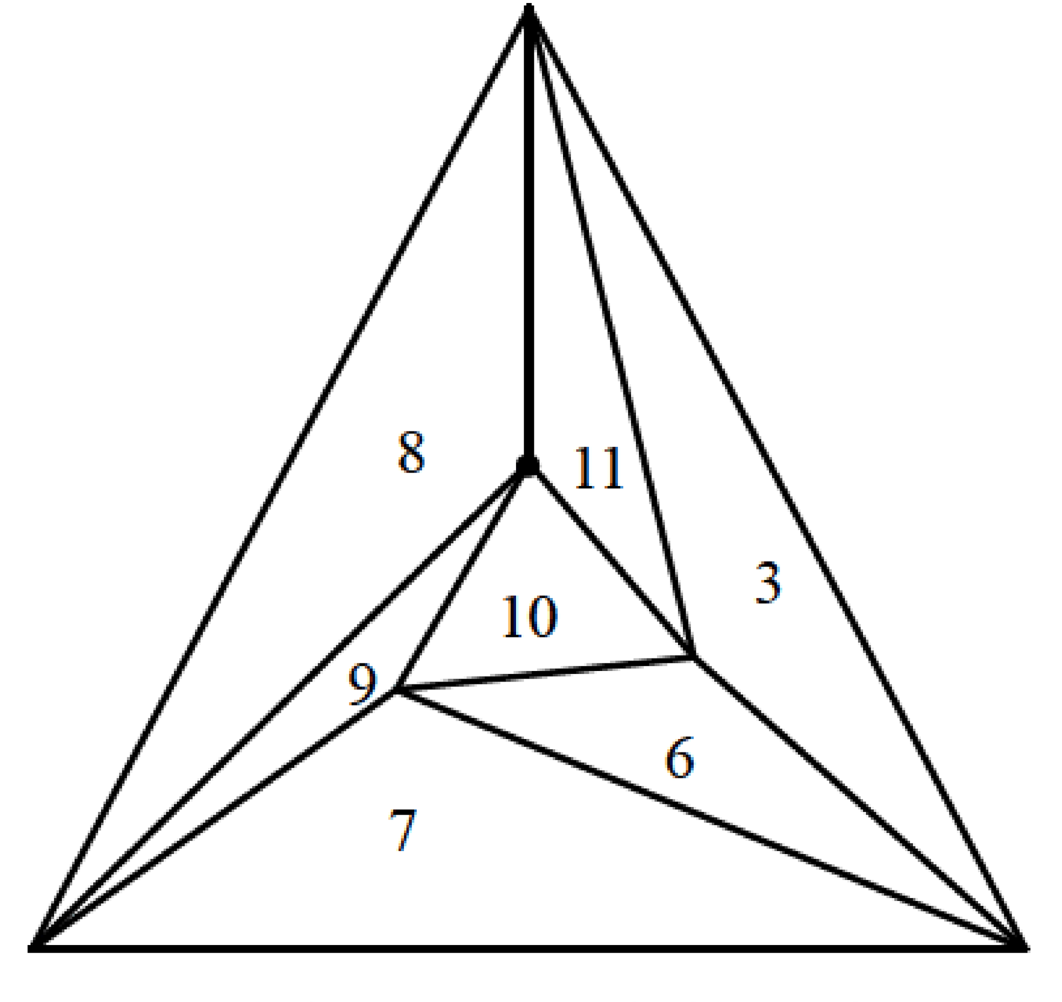
（7） }



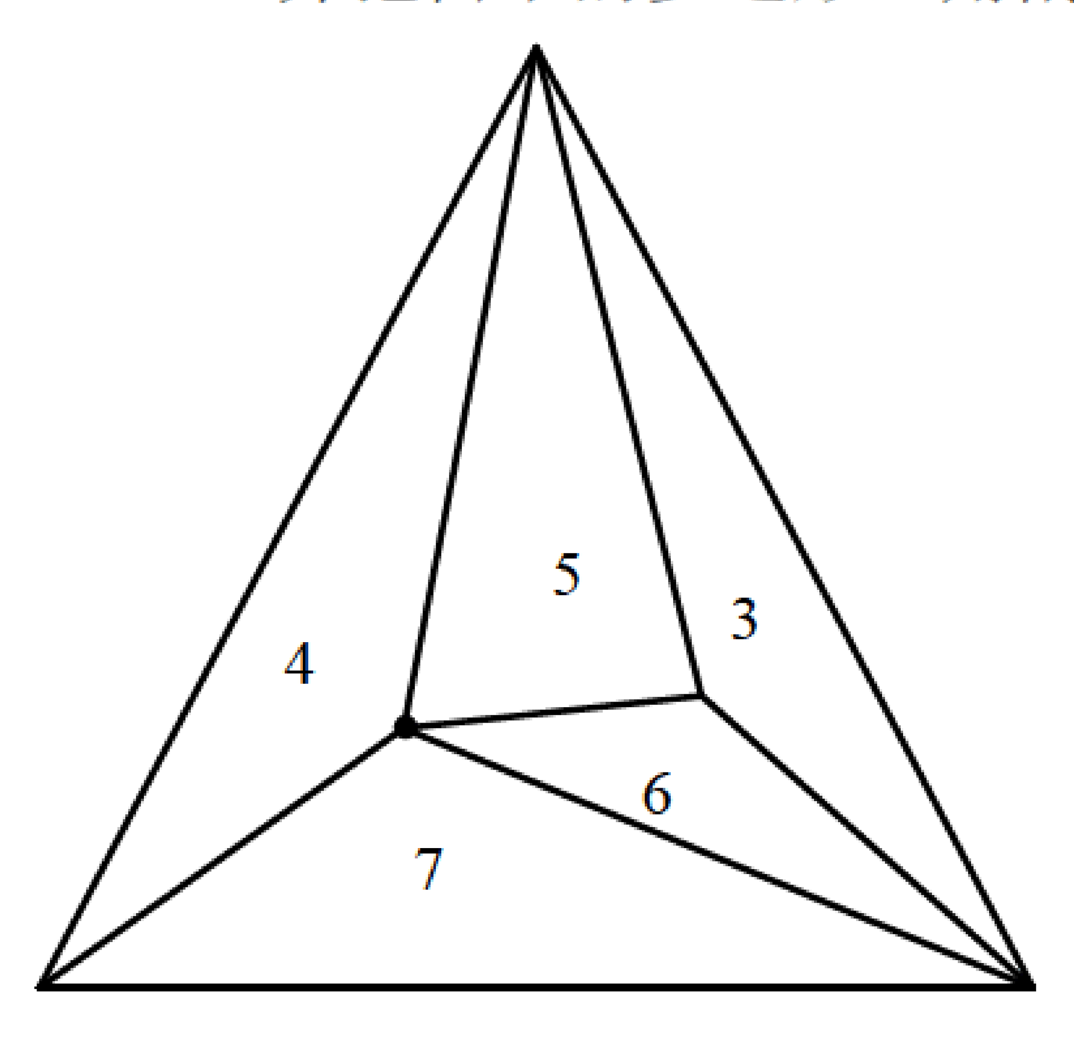
(a) 原图



(b) 删除了（a）中的黑点并把留下的多边形三角剖分

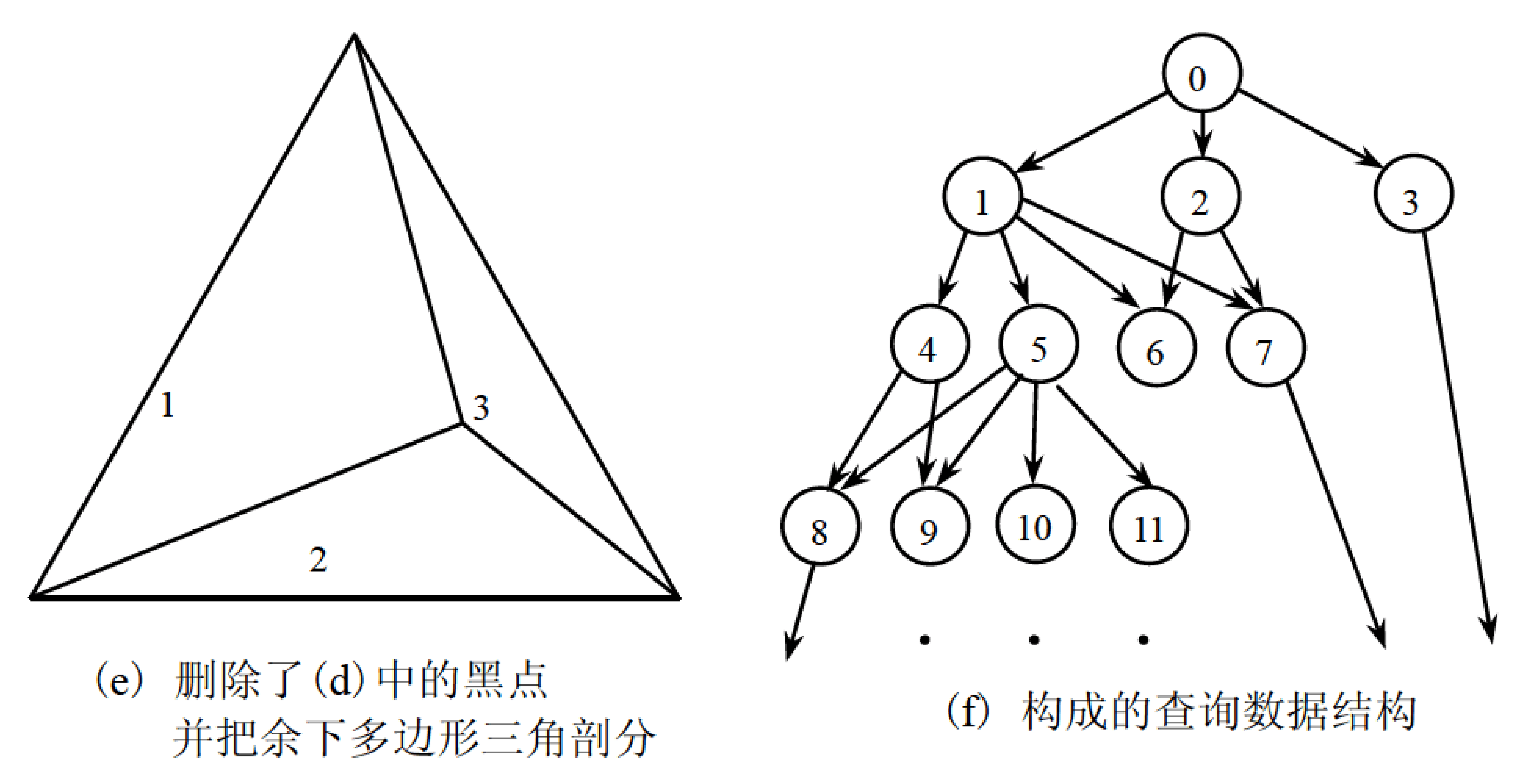


(c) 删除了（b）中的黑点并把留下的多边形三角剖分



(d) 删除了（c）中的黑点并把余下多边形三角剖分

图 4.14　基于 CDT 构造点定位查询数据结构



(e) 删除了(d)中的黑点并把余下多边形三角剖分

(f) 构成的查询数据结构

图 4.14 （续）

图 4.14 显示了算法预处理查询数据结构的生成过程。从图 4.14（a）到图 4.14（b），从图 4.14（b）到图 4.14（c），从图 4.14（c）到图 4.14（d），从图 4.14（d）到图 4.14（e），重复地执行了算法 4.3 循环体内（第 3~6 行）的操作。

在图 4.14 中，为了图的简化和说明清楚，最后图 4.14（e）中只有三个三角形时循环才结束，而不是按算法 4.3 中的方法，循环到余下不超过八个三角形便结束。图 4.14（f）是算法 4.3 建立的数据结构。该结构的每一个结点是一个三角形。这结构虽然不是一棵树，如果由最高的“0”结点到最下面的结点的深度为 $O(\log n)$，以及每个中间结点指出的指针数不超过一个和 n 无关的常数，则用对树相同的搜索算法，可在 $O(\log n)$ 时间内为询问点定位，其中 n 是点集 P 中的点数。

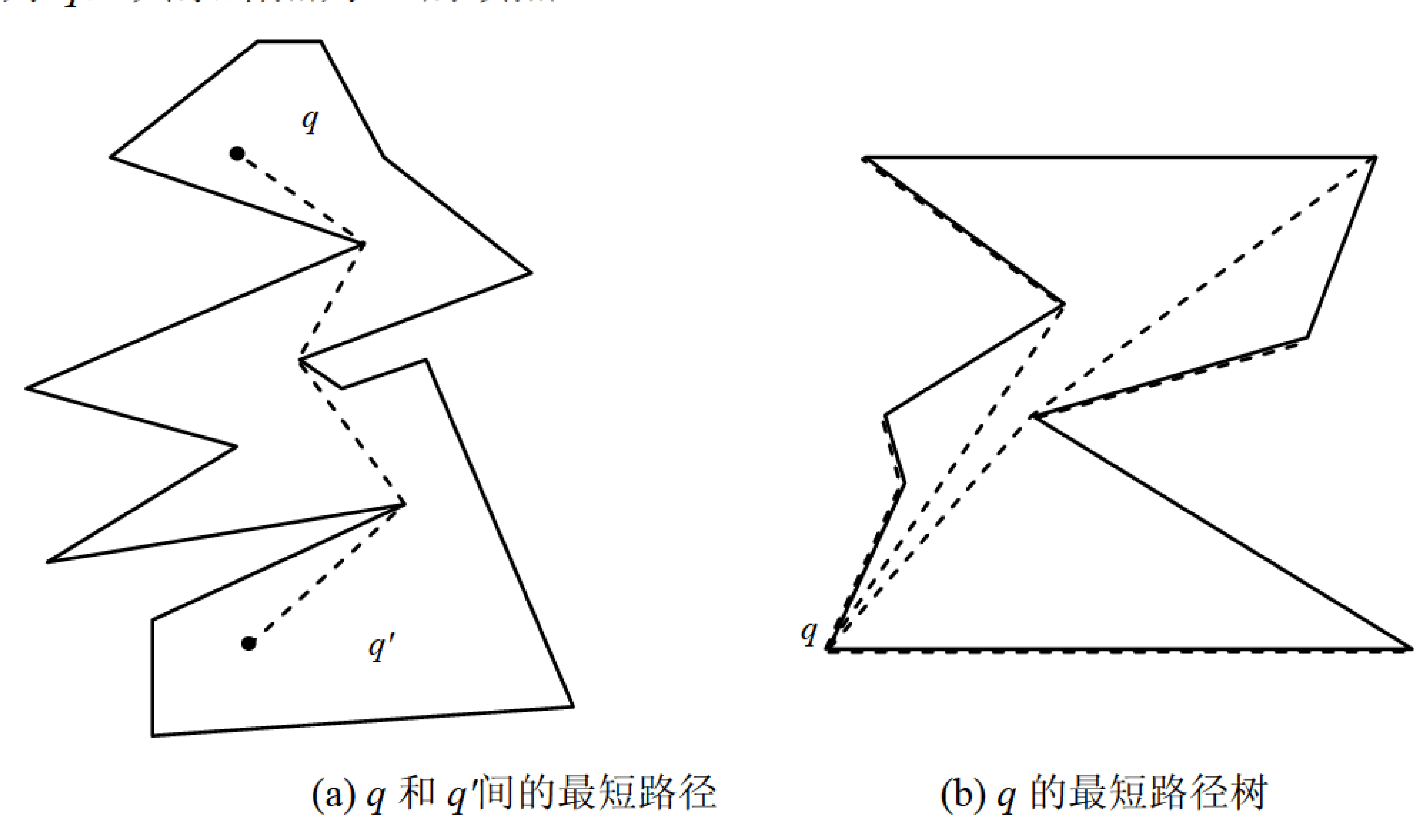
4.3.4 简单多边形中的最短路径与可见性计算

最短路径与可见性计算是计算几何的经典研究内容，应用十分广泛。Guibas 介绍了简单多边形内计算最短路径、点的可见性、线段可见性的方法 [Guibas1987]。这些方法需要在预处理阶段构建最短路径树。最短路径树可以基于多边形的 CDT 进行构建。

1. 最短路径树与最短路径

设 P 是有 n 个顶点的简单多边形，点 q 和点 q' 是 P 中的两点。在 P 中 q

和 q'之间的欧几里得最短路径是一条连接 q 和 q'的路径：该路径位于 P 内，且是连接 q 和 q'的所有路径中长度最短的，表示为π (q，q')，其结点是 q、q'和 P 的某些凹顶点（如图 4.15（a）所示）。从点 q 到 P 的每个顶点的欧几里得最短路径的集合组成了一棵以 q 为根的树，这棵树也被称为点 q 在 P 中的最短路径树，记作$\Pi(q)$（如图 4.15（b）所示）。$\Pi(q)$有 n 个结点，根结点为 q，其余结点为 P 的顶点。



(a) q 和 q'间的最短路径　　(b) q 的最短路径树

图 4.15　最短路径与最短路径树

下面为了叙述简便，我们不妨设 q 是 P 的一个顶点，介绍$\Pi(q)$的构造方法。对于 q 在 P 内部的情况，稍作修改即可得到$\Pi(q)$。

令 T 表示 P 的三角剖分的对偶图，其中 T 的结点是三角剖分中的三角形，如果两三角形相邻，即共享一条边，则它们在 T 中对应的结点之间有一条弧。显然，T 是一棵树，树的每个结点的度数最多是 3。因此，对于 P 的任意一个顶点 t，从 t 到 P 的三角剖分的某个三角形中的任意点 s，在 T 中存在唯一的路径。这条路径中的每条边 ab（a、b 是两个结点）则对应了 P 的一条对角线（该对角线是 a、b 分别对应的两个三角形的共享边）。这条路径依次穿过系列 P 的对角线：d_1，d_2，…，d_k。每条对角线 $d_i(1\leqslant i\leqslant k)$将 P 分为两部分，一部分包含 s，一部分包含 t。因此，T 中 s、t 之间的这条最短路径必然穿过

对角线 d_i 且只穿过一次。

令 $d=uw$ 是 P 的一条对角线或边，且 a 是 u、w 在 $\Pi(q)$ 中的最深的共同祖先（注：a、u、w 都是 P 的顶点，也是 $\Pi(q)$ 中的结点）。显然，$\pi(a, u)$ 和 $\pi(a, w)$ 都是向外凸的。我们称 $F=\pi(a, u)\cup\pi(a、w)$ 形成一个关于对角线 $d=uw$ 的漏斗（如图 4.16 所示），点 a 称为该漏斗的尖头。

假设 $d=uw$ 是 DT(P)中用到 P 的一条对角线，$\triangle uwx$ 是 DT(P)中的一个三角形，uw 是其一条边，且$\triangle uwx$ 与 F 和 d 之间围成的区域无交（如图 4.16 所示）。可以肯定，$\pi(q, a)$必是$\pi(q, x)$的一部分，其余的部分是$\pi(a, x)$。而$\pi(a, x)$的一部分在$\pi(a, u)$或$\pi(a, w)$上，另一部分在 F 和 d 之间围成的区域内；或$\pi(a, x)=ax$。图 4.16 中，$\pi(a, v)$在$\pi(a, u)$上，vx 在 F 和 d 之间围成的区域内。这里，vx 与$\pi(a, u)$相切于 v 点。基于这种观察，可设计一种计算 $\Pi(q)$ 的算法。

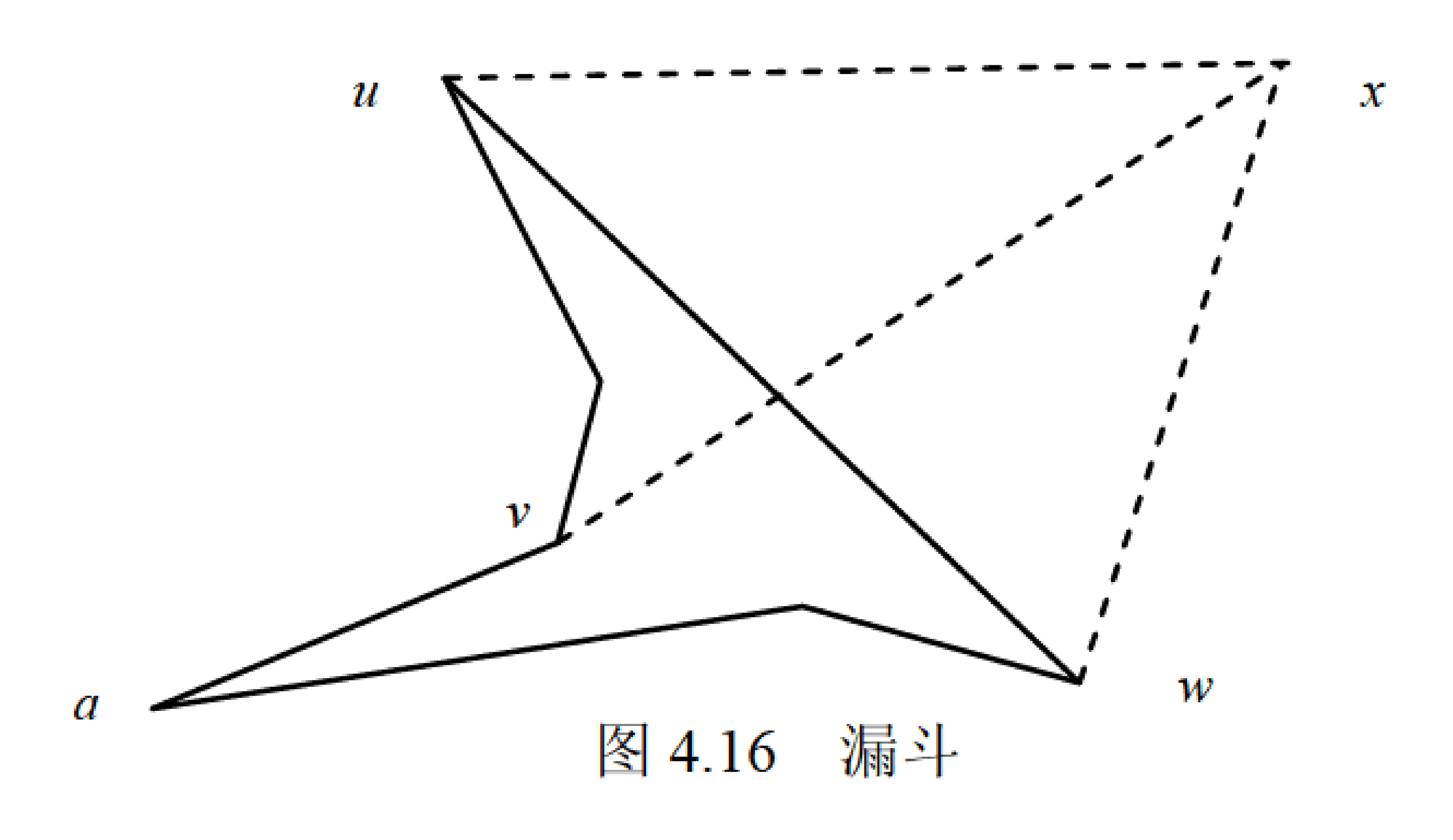


图 4.16　漏斗

在 DT(P)中，当计算到 P 的对角线 uw 时，设算法维护的当前的漏斗为 $F=F_{uw}$，用有序链表表示：$F=[u_l, u_{l-1}, \ldots, u_1, a, w_1, \ldots, w_k]$，其中 $a=u_0=w_0$ 是 F 的尖头；$\pi(a,u)=[u_0, u_1, \ldots, u_l]$，$\pi(a, w)=[w_0, w_1, \ldots, w_k]$。这里，$u=u_l$，$w=w_k$。用 CUSP($F$)表示 F 的尖头 a。

算法 4.4 PATH(F)　计算最短路径树

//令 u 和 w 表示 F 的链表中第一个和最后一个元素，a=CUSP(F)表示 F

//的尖头，这样，　$F=\pi(a, u)\cup\pi(a, w)$

//令△uwx 为 DT(P)的唯一的包含边 uw 且未被处理过的三角形

（1）在 DT(P)中查找包含边 uw 且未被处理过的三角形。

（2）如果没找到，返回。

（3）否则，设其为△uwx，x 为 P 的顶点。

（4）在 F 中查找点 v，满足 vx 与 F 相切于 v。如果切点不存在，则 $v=a$。

（5）将 $F\cup\{x\}$ 分成两个新漏斗：$F_1=[u, \ldots, v, x]$，$F_2=[x, v, \ldots, w]$。

（6）如果 v 在$\pi(a, u)$上，则 CUSP(F_1)=v，CUSP(F_2)=a。

（7）否则，CUSP(F_1)=a，CUSP(F_2)=v。

（8）$\pi(q, x)=\pi(q, v)\cup vx$。输出$\pi(q, x)$。

（9）如果线段 ux 是 P 的对角线，则递归调用 PATH(F_1)。

（10）如果线段 wx 是 P 的对角线，则递归调用 PATH(F_2)。

（11）返回。

在初始调用 PATH(F)时，如果 q 只是 DT(P)的一个三角形的顶点，令 u，w 分别取该三角形的另外两个顶点，且令 CUSP(F)=q。否则，需要逐次处理以 q 为顶点的三角形，分别设置 F 并调用 PATH(F)。

2. 最短路径查询

通过算法 4.4，可计算出$\Pi(q)$。在对 P 进行约束三角剖分的基础上，算法时间复杂度为 $O(n)$。基于$\Pi(q)$，对于 P 的任意顶点 v，可在 $O(\log n)$时间内查出$\pi(q, v)$。

对上述算法稍作修改，可处理更一般情况：对于多边形 P 内一点 q，在预先计算出其最短路径树的情况下，可在 $O(\log n)$时间内查出其到多边形 P 内任一点 p 的最短路径。这里不再赘述。

3. 点的可见性计算

在 3.3.4 节介绍了一种基于复杂多边形的 Voronoi 图的点可见性计算方法。这里，对于一个简单多边形，我们在计算 $\Pi(q)$ 的基础上，可以在 $O(n)$ 时间计算出 q 的可见多边形。通过按顺时针依次遍历 P 的每条边的方式，计算该边相对于 q 的可见部分。对于 P 的一条边 uw，首先在 $\Pi(q)$ 查出 $\pi(q, u)$ 或 $\pi(q, w)$。令 qu_1、qw_1 分别是 $\pi(q, u)$ 或 $\pi(q, w)$ 的第一条边，如果 $qu_1=qw_1$，则 uw 相对于 q 不可见；否则，分别计算线段 uw 和射线 qu_1、qw_1 之间的交点，交点之间的部分即是 uw 相对于 q 的可见部分。

如图 4.17 所示，uw 相对于 a 的可见部分是 uw 与 av，az 交点之间部分，这里，av 和 az 分别是 $\pi(a, u)$ 和 $\pi(a, w)$ 的第一条边。

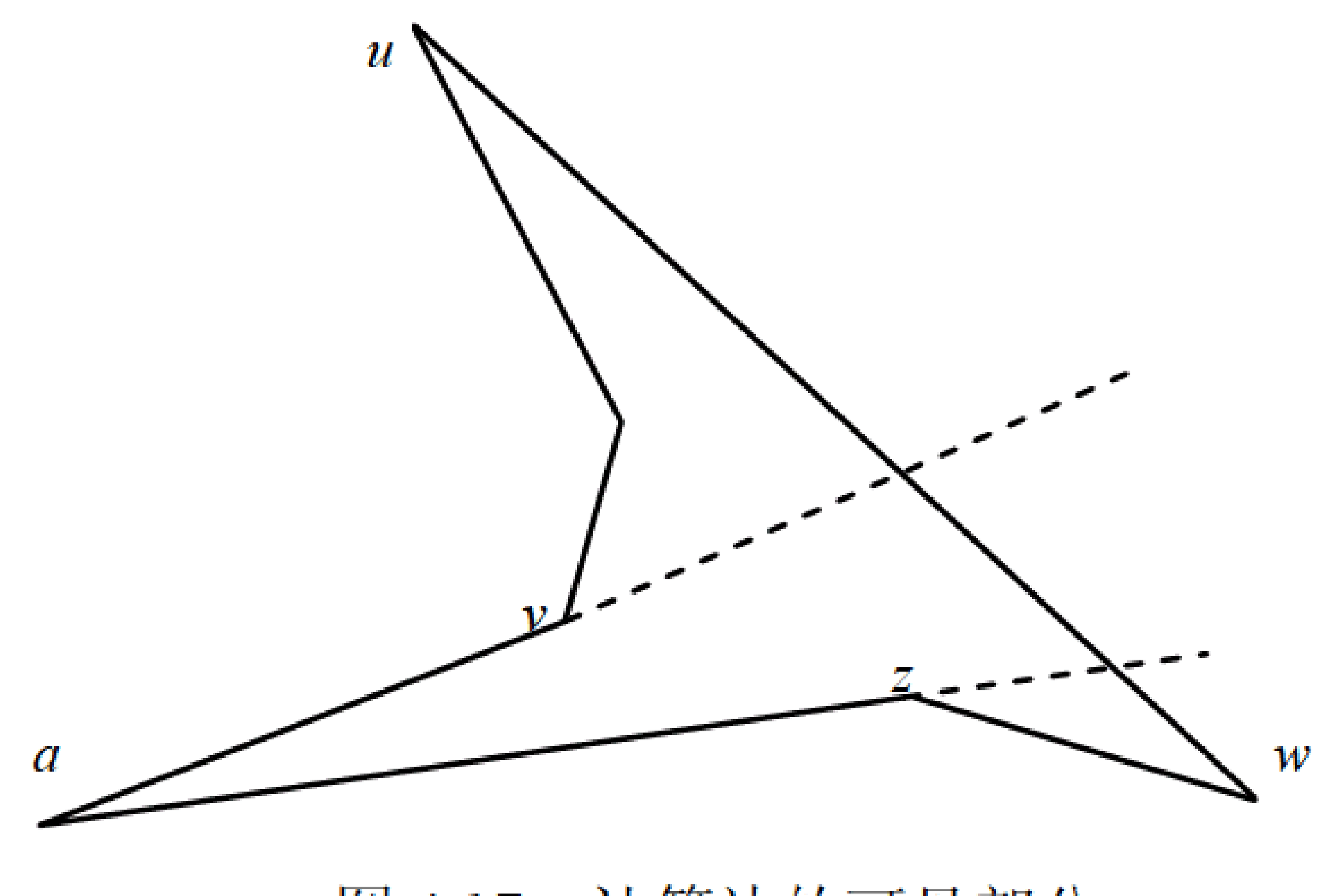


图 4.17 计算边的可见部分

4. 线段的弱可见性计算

对于一个多边形 P 和其中的线段 L，如果 L 中存在一点可以看到 P 中某点 q，则称点 q 相对于 L 是弱可见的。L 的所有弱可见点的集合就叫做 L 的弱可见多边形。在警卫巡逻、机器人路径规划、光照计算等应用方面，常需要计算线段的弱可见多边形以确定警卫、机器人观察或光照覆盖的区域。这里介绍一种基于最短路径树的线段的弱可见多边形计算方法。该算法只适于简单多边形。

算法首先将简单多边形中的 pq 延长后将该多边形分割成两个子多边形，然后分别计算 pq 在两个子多边形中的可见多边形，最后合并得到 pq 在原多

边形中的可见多边形。算法在预处理中对该多边形进行约束三角剖分，然后分别构造出 pq 的两个端点 p、q 的最短路径树。对于 pq 在每个子多边形 P 中的可见多边形的计算，主要是基于最短路径树计算 P 的每条边上相对于 pq 弱可见的部分。

利用最短路径树，可以判断一条边相对于另一条边是否可见，以及哪部分可见。一般地，一个多边形 P 中两点 q 和 q'之间的最短路径可分为四类：只向左转（如图 4.18 中的$\pi(p_1, p_7)$，粗虚线所示），直接连接（如图 4.18 中的$\pi(p, q)$，粗虚线所示），只向右转（如图 4.18 中的$\pi(q_1, q_9)$，粗虚线所示），既左转又右转（如图 4.15（a）中的$\pi(q, q')$，虚线所示）。

对于一个简单多边形 P 的两条边$e_i = p_i p_{i+1}$和$e_j = p_j p_{j+1}$，p_i、p_{i+1}、p_j和p_{j+1}是以多边形内部在前进方向的左侧顺序遍历 P 的边界时依次出现的。假定已有最短路径树$\Pi(p_i)$和$\Pi(p_{i+1})$。通过观察不难发现，边e_j中含有一点从e_i可见，当且仅当最短路径$\pi(p_i, p_{j+1})$和$\pi(p_{i+1}, p_j)$都是向多边形内部凸的。如图 4.18 中$e_i = p_1 q_1$、$e_j = q_9 p_7$所示，$\pi(p_1, p_7)$和$\pi(q_1, q_9)$（粗虚线）都是向多边形内部凸的。这些路径向内凸当且仅当$\pi(p_i, p_{j+1})$只向左转，并且$\pi(p_{i+1}, p_j)$只向右转。

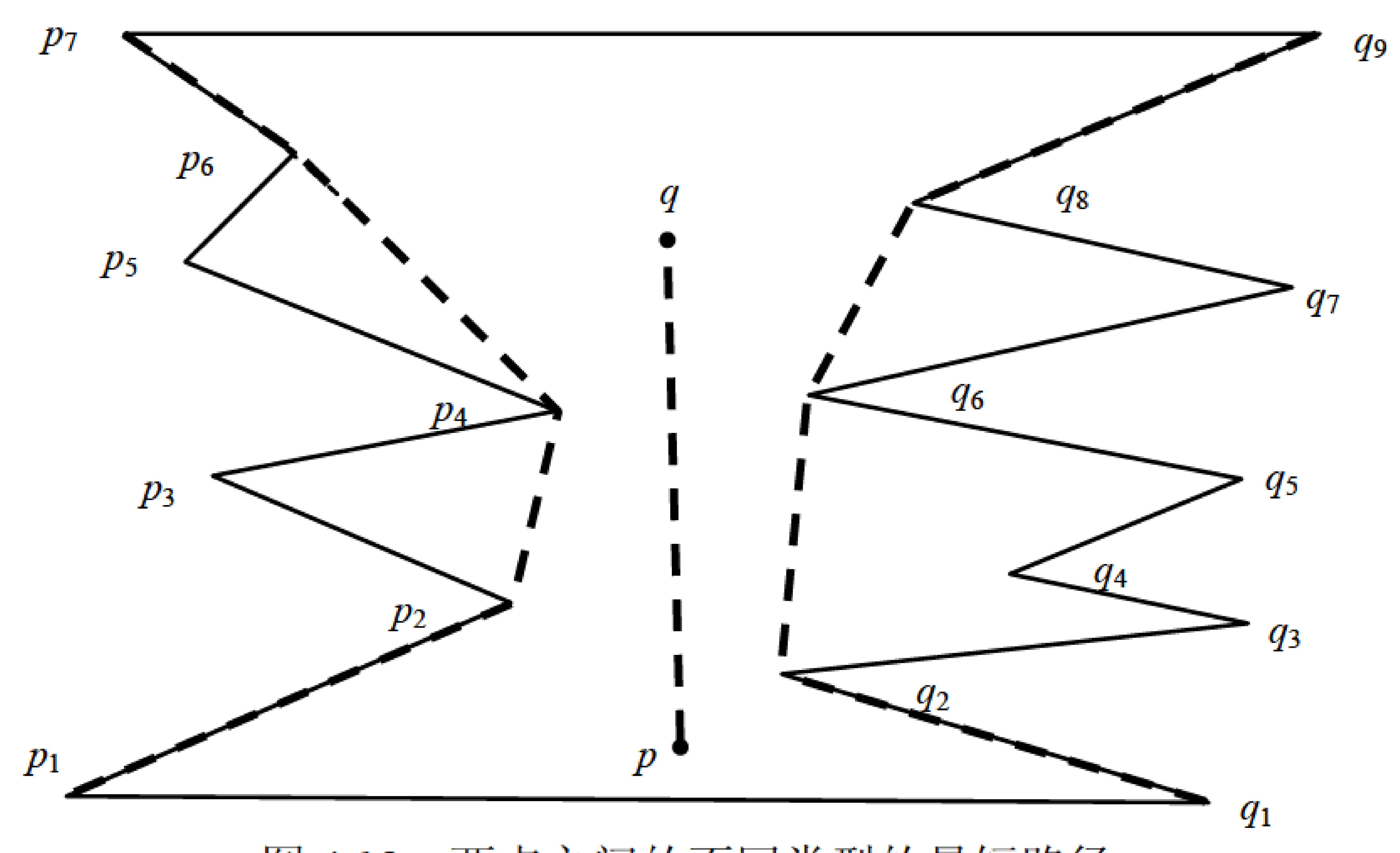


图 4.18　两点之间的不同类型的最短路径

因此，我们得到一种计算边$e_j = p_j p_{j+1}$中相对于边$e_i = p_i p_{i+1}$的弱可见的方法，步骤如下。

（1）首先，从最短路径树$\Pi(p_i)$和$\Pi(p_{i+1})$找到最短路径$\pi(p_i, p_{j+1})$和$\pi(p_{i+1}, p_j)$。

（2）如果$\pi(p_i, p_{j+1})$或$\pi(p_{i+1}, p_j)$不是向多边形内部凸的（这可通过检测路径是否只向左转或只向右转即可得知），则可断定边$e_j = p_j p_{j+1}$中没有任何点相对于边$e_i = p_i p_{i+1}$是弱可见的（如图 4.19 所示）。

（3）否则，计算$\pi(p_i, p_{j+1})$和$\pi(p_{i+1}, p_j)$的两条公共切线（如图 4.20 中的两条虚线）。这些切线与$e_j = p_j p_{j+1}$的交点间的部分即为相对于边$e_i = p_i p_{i+1}$的弱可见部分（如图 4.20 中的$r_1 r_2$为边$e_j = q_9 p_7$上相对于边$e_i = p_1 q_1$的弱可见部分）。

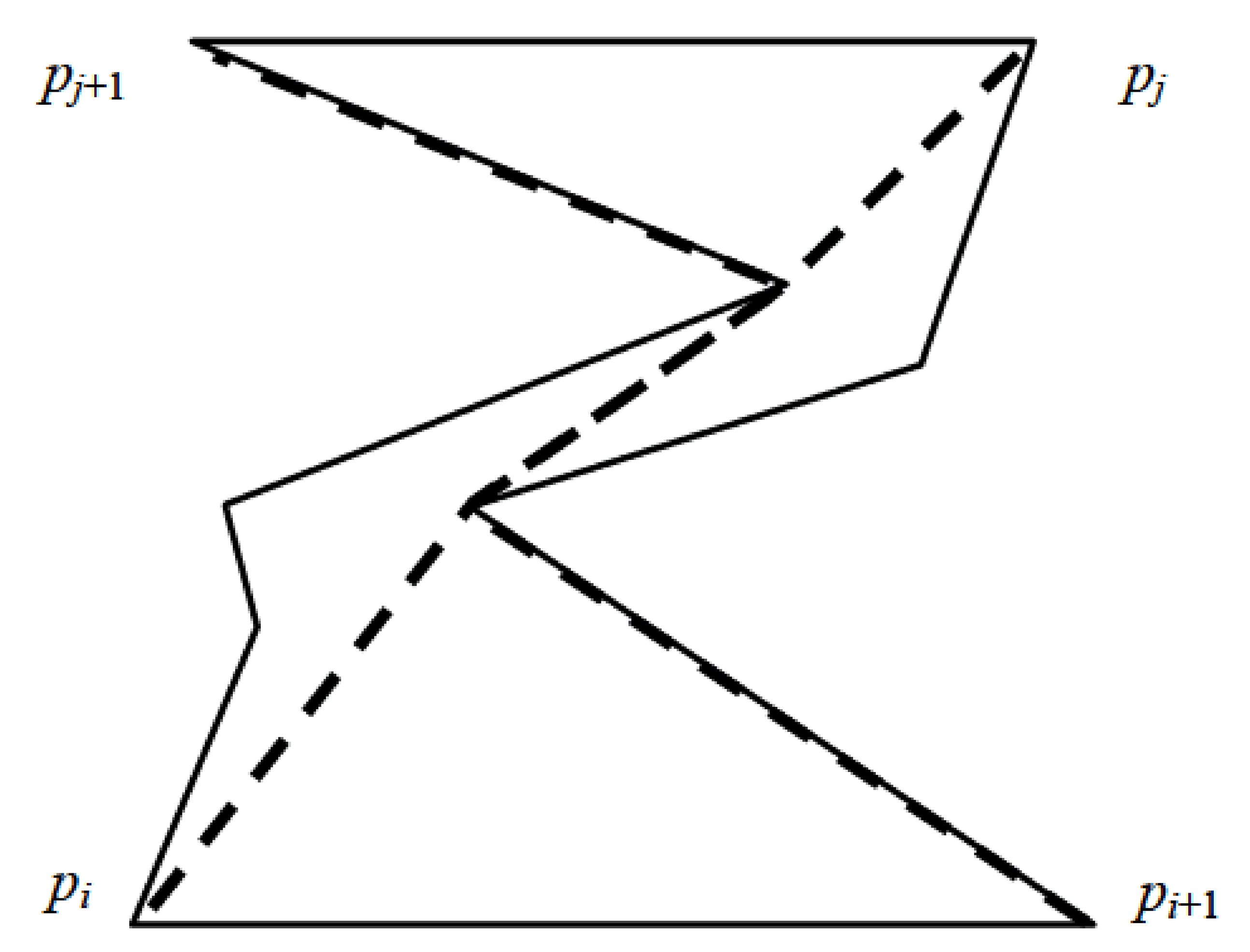


图 4.19　最短路径$\pi(p_i, p_{j+1})$和$\pi(p_{i+1}, p_j)$是非凸的

总之，对于简单多边形P中输入的任一条查询边ab，其弱可见多边形的计算方法如下。

（1）对简单多边形P进行约束三角剖分。

（2）计算a、b的最短路径树。

（3）延长ab将P分割成两个子多边形P_1，P_2。

（4）我们按多边形P顶点编号的顺序遍历P_1的边，从顶点p_{i+1}开始，将遍历到的每条边的可见部分依次保存到一个链表中，链表中保存的是该可见部分的两个端点。每条边的可见部分的计算用上面介绍的方法。

如果在链表中出现了一个端点被连续重复记录的现象，则这个端点是从 ab 可见的 P 的一个顶点。去掉这种重复，最后可获得 P_1 中边 ab 的弱可见多边形，链表中记录的就是该弱可见多边形的顶点。

（5）用类似步骤（4）的方法计算 P_2 中 ab 的弱可见部分。

（6）P_1 和 P_2 中 ab 的弱可见部分的并集即得到 P 中 ab 的弱可见部分。

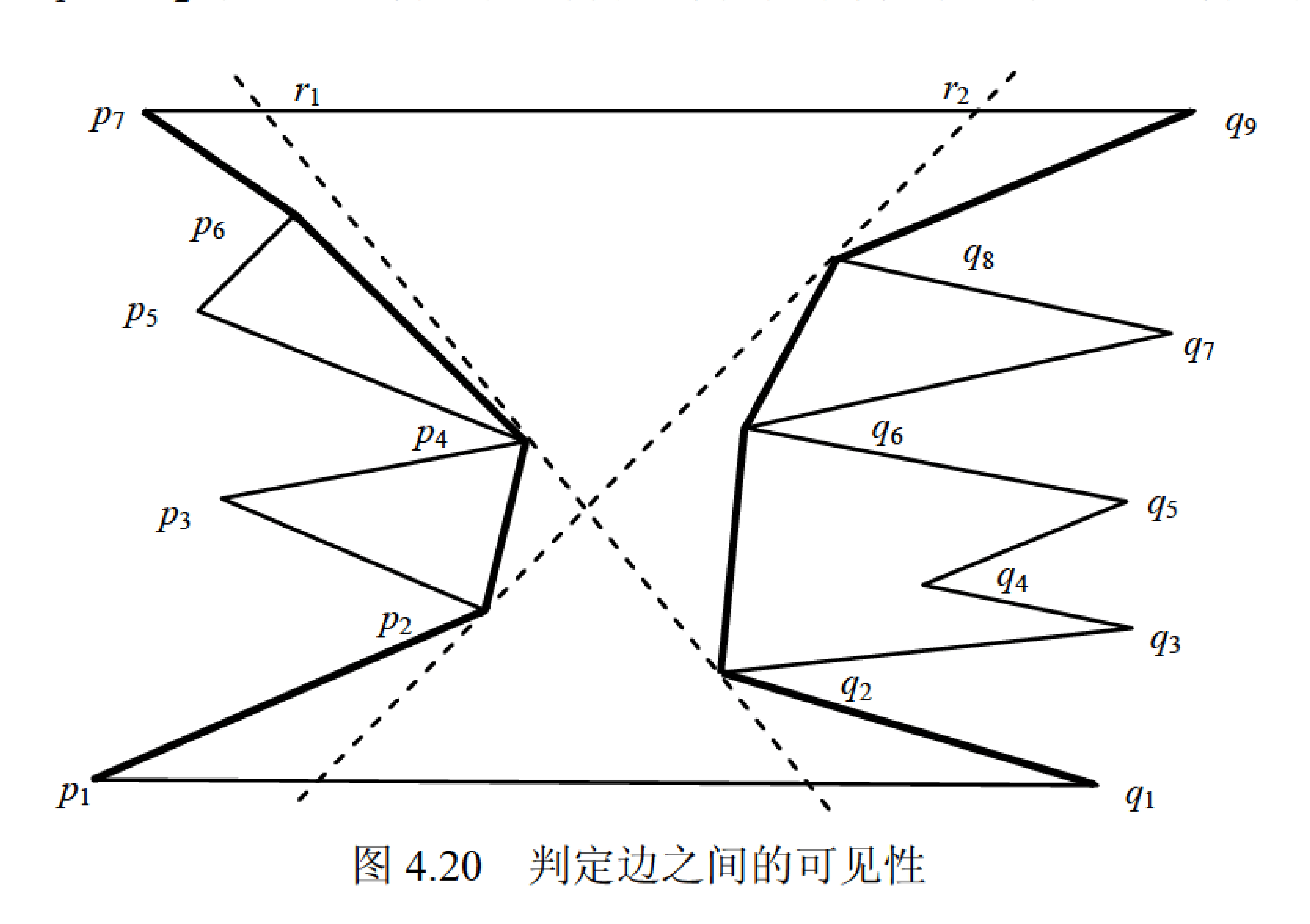


图 4.20　判定边之间的可见性

4.3.5　复杂多边形中的可见性计算

本节介绍一种计算复杂多边形内的点的可见性算法[Lu2011]。该算法首先对多边形P进行约束Delaunay三角剖分，将P分割成$O(h)$个简单多边形；然后将每个简单多边形的内部和外部分割成可见单元，并建立数据结构。基于此数据结构可以在$O(|\mathrm{VP}(P, v)|+ h+ \log^2 m+ h\log(n/h))$时间内报告$\mathrm{VP}(P, v)$（其中$m<n$），只需要预处理时间$O(n^2\log n)$和空间$O(n^2)$。

1. 多边形分割

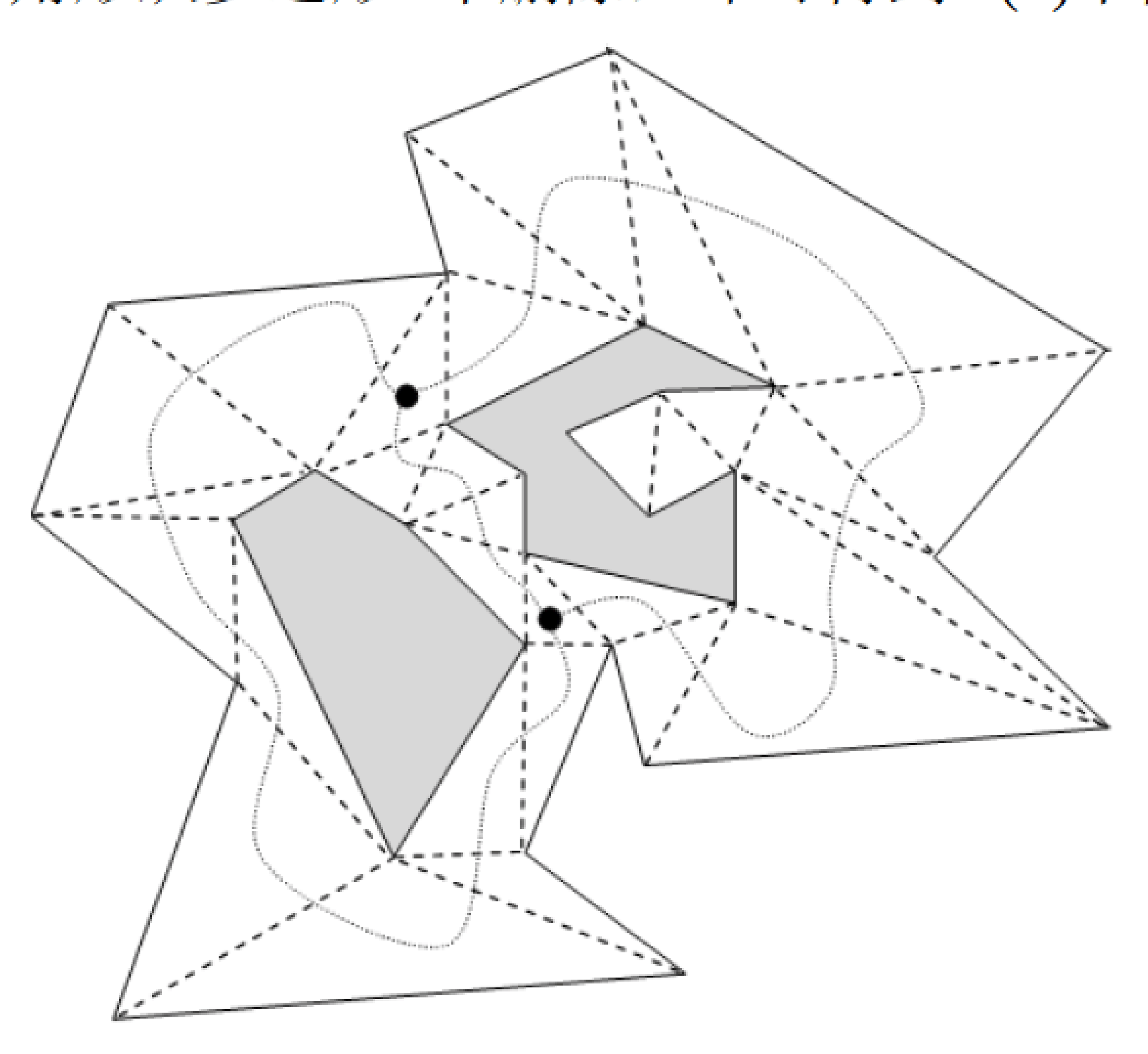
假设多边形P的洞数$h\geqslant 2$，将多边形P分割成简单多边形的算法如下。

（1）计算CDT(P)，并计算其对偶图T（见图4.21（a））。

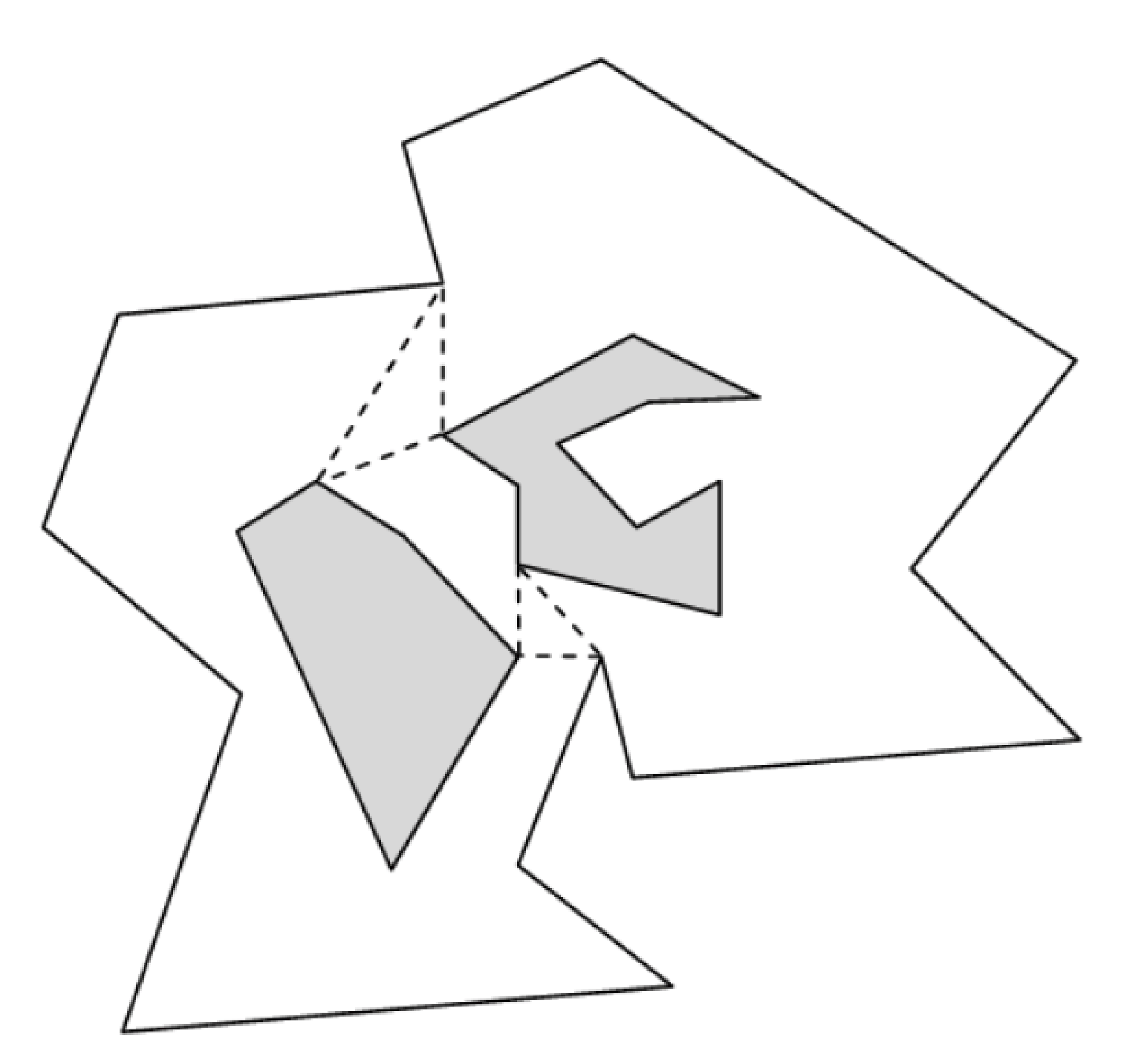
（2）寻找连接三角形。首先，删除T中度为1的顶点及其关联的边，并重

复此操作直至T中不存在度为1的顶点；然后，删除T中度为2的顶点，并将其关联的两条边替换为一条边；最后，得到一个有$h+1$个面、$2h-2$个顶点和$3h-3$条弧的图。此图中的结点对应CDT(P)中的一个三角形，我们称这$O(h)$个三角形为连接三角形。

（3）将连接三角形从多边形P中删除，即可得到$O(h)$个简单多边形。



(a) 多边形的 CDT，两个黑点所在三角形是连接三角形



(b) 多边形被分割为三个简单多边形

图 4.21　分割多边形

我们将连接三角形的边也称为“切割对角线”。对于任一个简单多边形，它的两个侧面都是多边形P的边界（见图4.21（b））。

2. 数据结构

将多边形分割为简单多边形后，点v的可见多边形VP(P，v)也被分割为简单多边形内的可见部分。例如，图4.22中的VP(P，v)即是VP(P_1，v)（网状区域）、VP(t_1，v)（斜线区域）和VP(P_2，v)（点状区域）的并集。我们可以将VP(P_1，v)看作是VP(P，v)的初始版本。为计算VP(P_1，v)，我们采用文献[Aronov2002]中的方法将P_1的内部做可见性分割并建立储存可见性信息的数据结构。对于其他简单多边形可以进行类似操作。

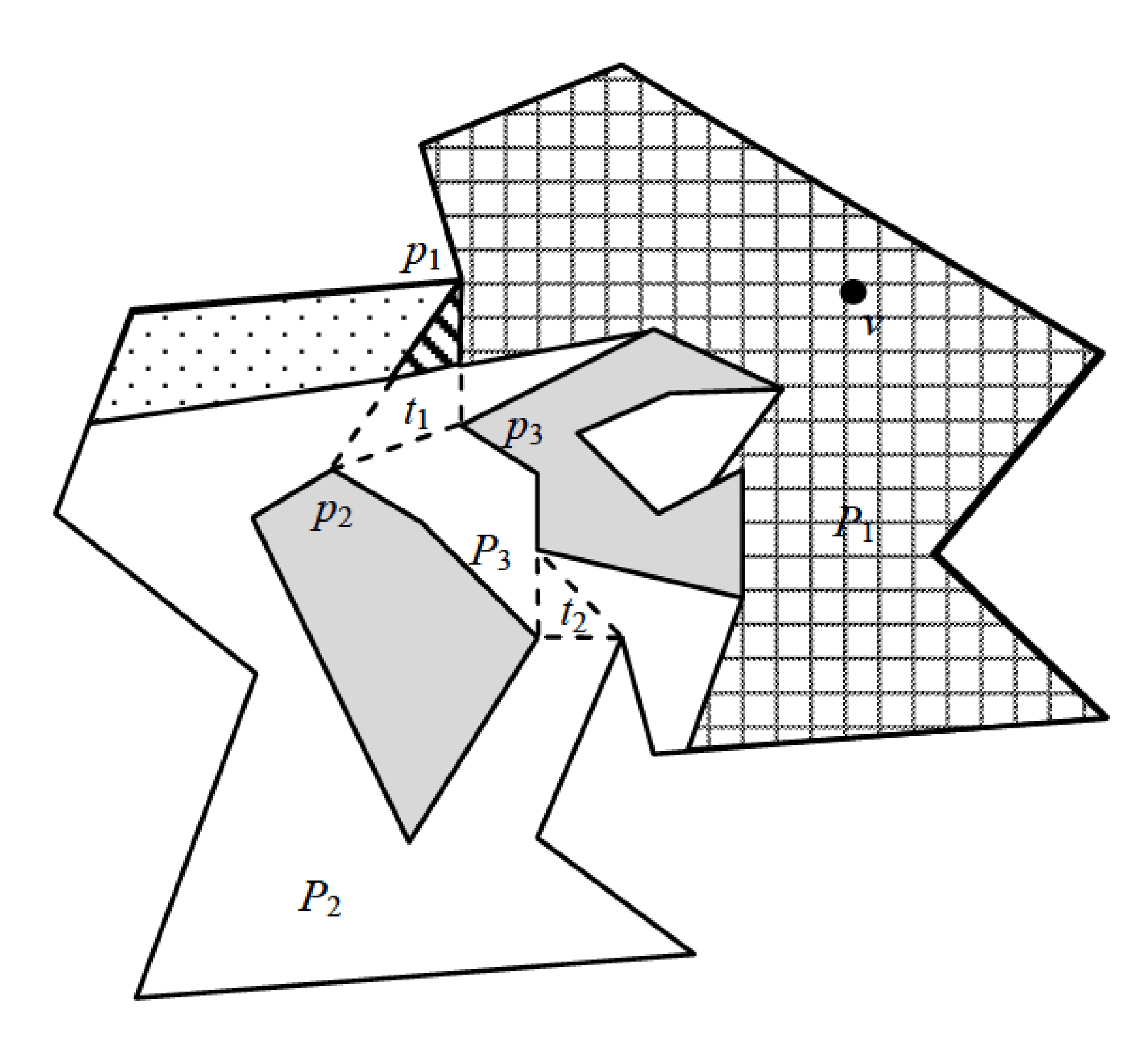


图 4.22　VP(P，v)是 VP(P_1，v)（网状区域）、VP(t_1，v)（斜线区域）和 VP(P_2，v)（点状区域）的并集

接下来，仍然应用文献[Aronov2002]中的方法计算VP(P_2，v)，对P_2的外部即边p_2p_1的右侧部分进行可见性分割（见图4.23）。根据文献[Aronov2002]，在定位了点v所在可见单元后，我们可以得到从点v经由线段$p_2\ p_1$的可见多边形VP$_e$(P_2，p_2p_1，v)。假设线段p_2p_1上的可见部分是q_1q_2，则VP(P_2，v)是VP$_e$(P_2，p_2p_1，v)与vq_2和vq_1所构成圆锥区域的交集（见图4.24）。类似的，对于边p_5p_4也需要做类似的操作。

作为结论，我们按照以下方式建立数据结构。

（1）对于每个简单多边形P_i，我们建立一个平衡三角剖分树的数据结构，并利用文献[Aronov2002]中的方法建立一个数据结构T_i，因此当查询点v位于P_i中时，可以得到VP(P_i，v)。

（2）对于每个简单多边形P_i和P_i的每个切割对角线e_j，我们分割其外部并利用文献[Aronov2002]中的方法建立一个数据结构，此数据结构支持当v不在P_i内但在P_i内仍存在v的可见点时，从点v经过e_j的可见多边形VP$_e$(P_i，e_j，v)的查询。显然，这个信息可以存储在T_i的根结点内。

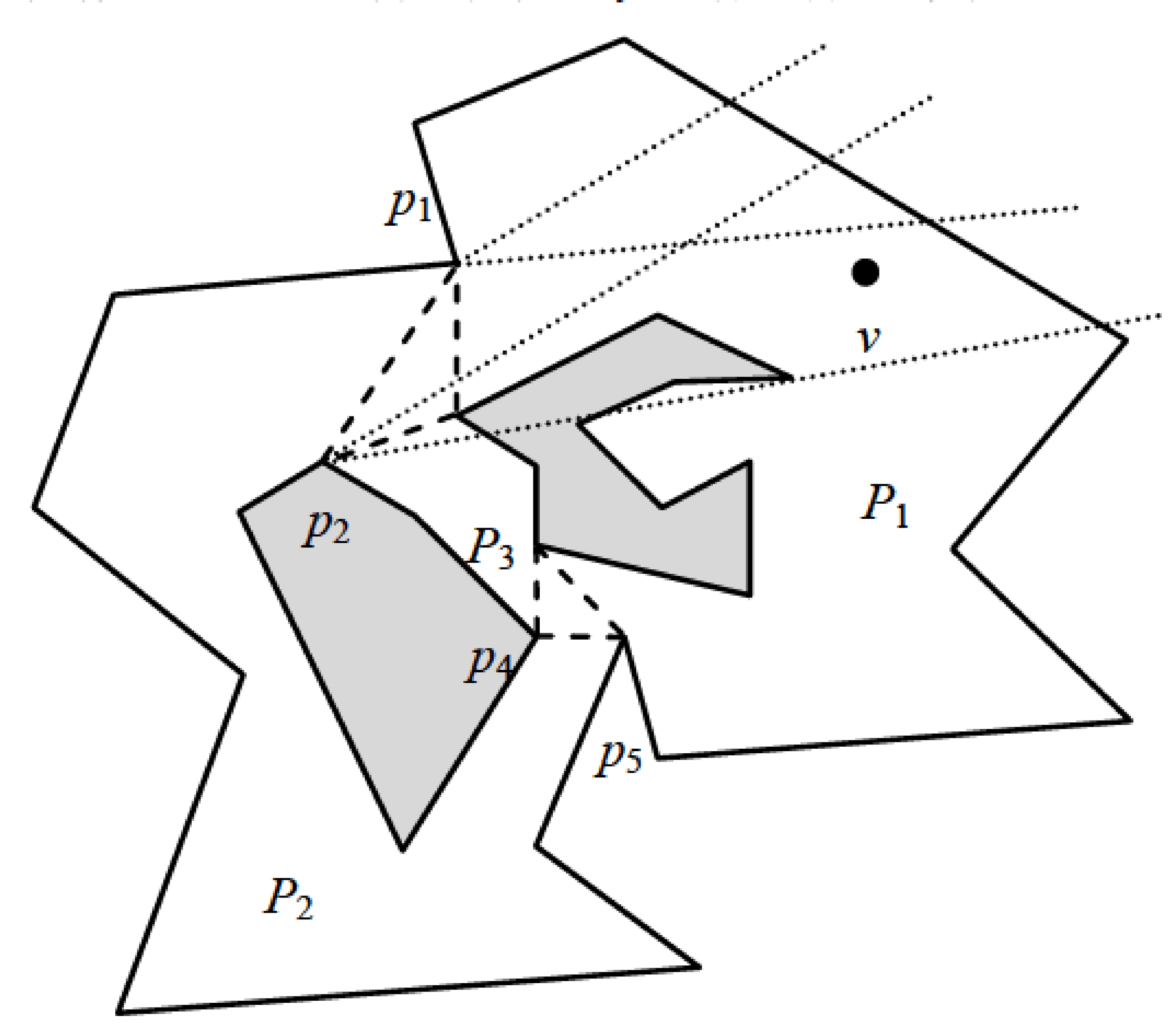


图 4.23　多边形 P_2 基于切割对角线 p_1p_2 的外部可见单元分割

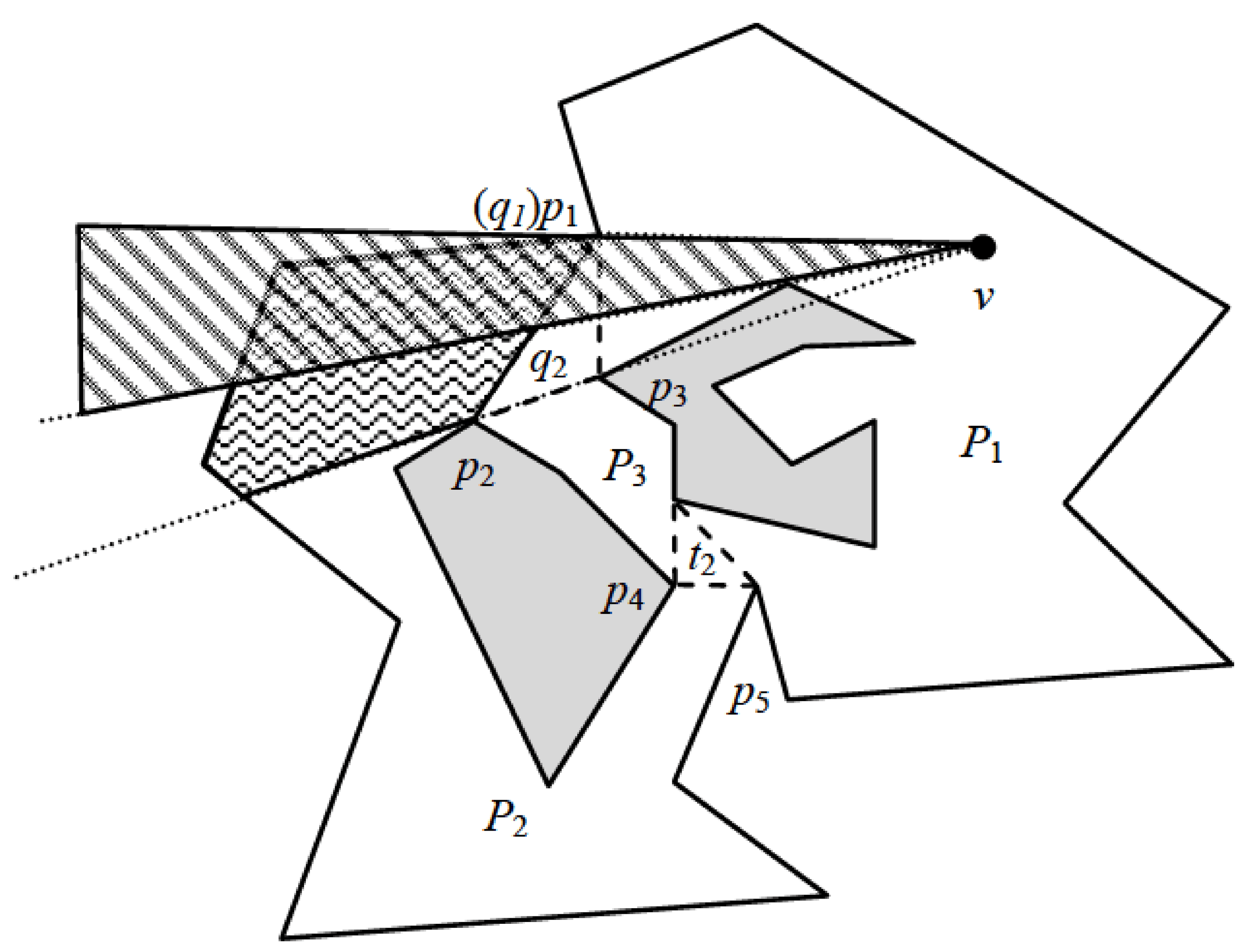


图 4.24　VP(P_2，v) 是 VP$_e$(P_2，p_2p_1，v)（波浪线区域）与 vq_2 和 vq_1 所构成圆锥（斜线区域）的交集

这里，需要对简单多边形建立一个平衡三角剖分树的数据结构。平衡三角剖分由Chazelle在1982年提出[Chazelle1982]。其基本思想为：对于一个简单多边形 P 而言，总是存在一个对角线 e，e 将 P 分为两部分，每部分最多 $2n/3$ 个顶点。基于这一观察，可通过迭代将多边形依次剖分为两个子部分，从而建立一棵平衡二叉树，其中每个结点 i 对应于一个子多边形 P_i 和 P_i 的对角线 e_i。结点 i 的左、右子树记为 L_i 和 R_i，通过沿 e_i 切割 P_i 得到。由于该平衡二叉树的叶结点都是三角形，故名平衡三角剖分树。

给定简单多边形 P 和任一查询点 q，将 P 按照上述方法剖分成 $O(\log n)$ 个不连接的子多边形即三角形，并将每个子多边形表示为平衡三角剖分树的一个结点，因此，在所有的子多边形中有且只有一个包含点 q。对于树中不同结点对应的每个子多边形 P'，计算其部分可见多边形，所有部分可见多边形的集合即为 VP(P，q)。

3. 可见性查询

在运行阶段，对于任一点 v，查询其可见多边形的算法如下。

算法 4.5　复杂多边形内的点的可见性查询

（1）确定 v 的位置；

（2）如果 v 在一个连接三角形 t_i 内，t_i 是 VP(P，v)是基础版本，转到第（4）步；

（3）否则，假设 v 在一个简单多边形 P_i 内，则利用 P_i 的内部数据结构计算 VP(P_i，v)，并作为 VP(P，v)的基础版本；

（4）对于 VP(P，v)的每一条落在连接三角形边上的可见边，记为 ab；

（4.1）将此连接三角形记为 t_i，将 VP(P，v)落在 t_i 上的边记为 e_i；

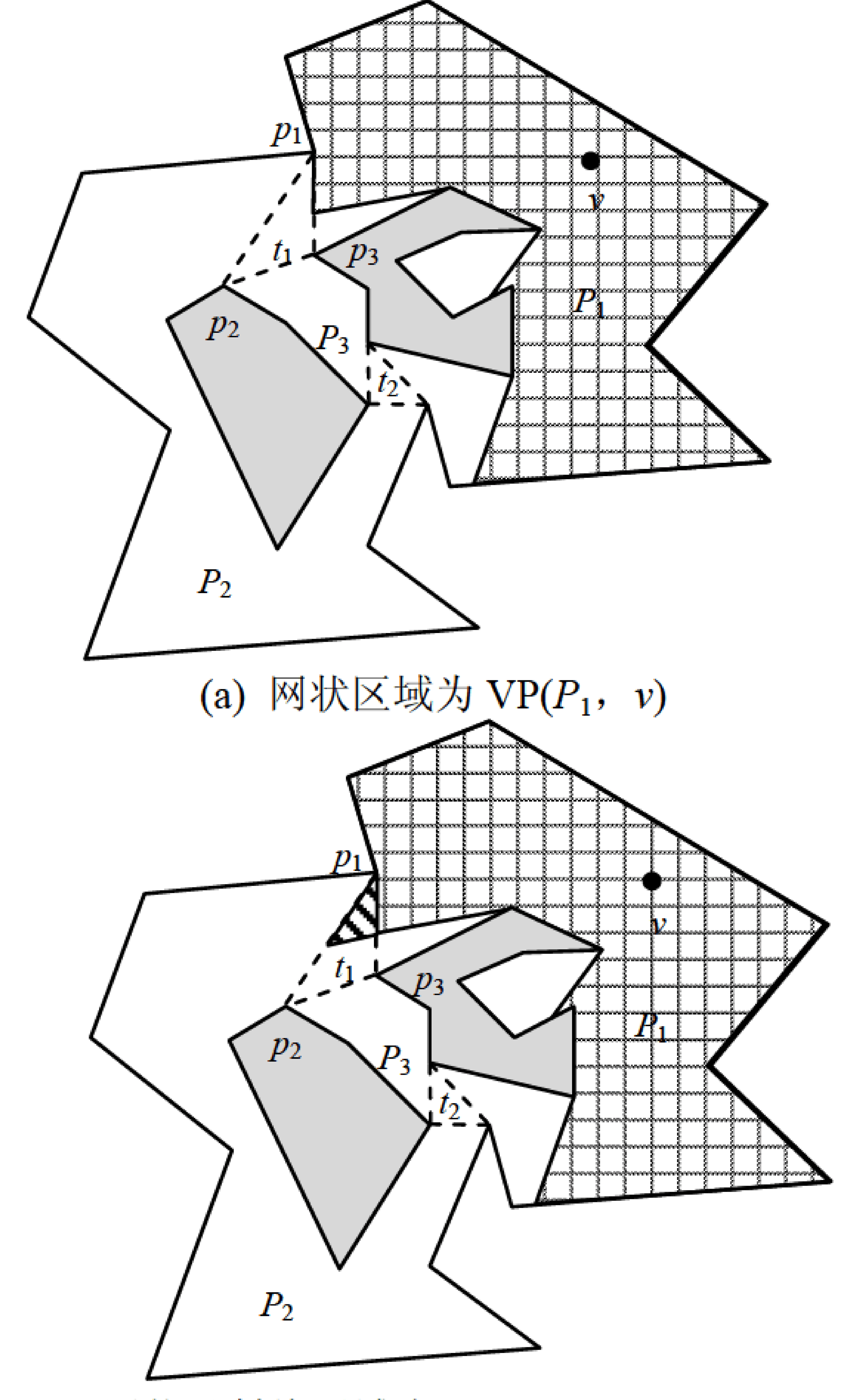
（4.2）计算 VP(t_i，v)，并将 ab 替换为 VP(t_i，v)，同时更新 VP(P，v)；

（4.3）如果 e_i 同时是某简单多边形 $P_j(i≠j)$的一条边，通过查询 P_j 关于 e_i 的外部数据结构计算 VP$_e$(P_j，e_i，v)。VP(P_j，v)是 VP$_e$(P_j，

e_i，v)与 va 和 vb 构成圆锥区域的交集。将 ab 替换为 VP(P_j，v)，同时更新 VP(P，v)；

（5）输出 VP(P，v)。

对于图 4.25 所示的例子，假定点 v 在多边形 P_1 内，则第（3）步得到 VP(P，v) = VP(P_1，v)（见图 4.25（a）），在第（4）步第一次迭代中，VP(P，v) = VP(P_1，v)∪VP(t_1，v)（见图 4.25（b）），第（4）步的最后一次迭代后，VP(P，v) = VP(P_1，v)∪VP(t_1，v) ∪VP(P_2，v)，其中 VP(P_2，v)是 VP$_e$(P_2，$p_2\,p_1$，v)与 vq_2 和 vq_1 所构成圆锥的交集，如图 4.22 所示。



(a) 网状区域为 VP(P_1，v)

(b) 网状及斜线区域为 VP(P_1，v)∪VP(t_1，v)

图 4.25　计算 v 的可见多边形的过程

在算法 4.5 中，第（1）步需要的时间是 $O(\log n)$，第（2）步所需时间是 $O(1)$，第（3）步可以在 $O(\log^2 n_i+|\mathrm{VP}(P_i, v)|)$时间内完成，第（4.2）步需要的时间是 $O(1)$，第（4.3）步需要的时间是 $O(\log n_j+|\mathrm{VP}(P_j, v)|)$，由于共有 $O(h)$个连接三角形，第（4）步在最坏情况下需要的时间是 $O(h+\Sigma\log n_j+|\mathrm{VP}(P, v)|)$，因此第（4）步所需的时间为 $O(h\log(n/h)+h+|\mathrm{VP}(P, v)|)$。因此，算法 4.5 的时间复杂度可写为 $O(\log^2 m+h\log(n/h)+h+|\mathrm{VP}(P, v)|)$，其中 m 表示点 v 所在简单多边形的点数，$m<n$。

值得注意的是，某条连接三角形的边可能对应多个可见区域，如图 4.26 所示例子中的加粗部分，P_2 关于 dj 的外部区域会被查询三次。我们采用以下方法避免此类重复查询所可能增加的算法复杂度。如图 4.26 中，$P_2 = dnmlkj$，$\mathrm{VP}(P_2, v)$是 $\mathrm{VP_e}(P_2, dj, v)$与 vd 和 ve、vf 和 vg、vh 和 vi 所构网状成三个圆锥区域的交集。只须在第一次遇到 dj 边时查询 $\mathrm{VP_e}(P_2, dj, v)$一次，此后，当计算完 $\mathrm{VP_e}(P_2, dj, v)$与 vd 和 ve 所构成圆锥区域的交集后，将得到的可见部分从 $\mathrm{VP_e}(P_2, dj, v)$中去掉，并保存剩下的部分，即 $\mathrm{VP_e}(P_2, e_j, v)$。当 dj 边第二次遇到时，只须计算 $\mathrm{VP_e}(P_2, e_j, v)$ 与 vf 和 vg 所构成圆锥区域的交集，以此类推。这部分操作并不影响算法的整体时间复杂度。算法 4.5 的全部流程参见图 4.27。

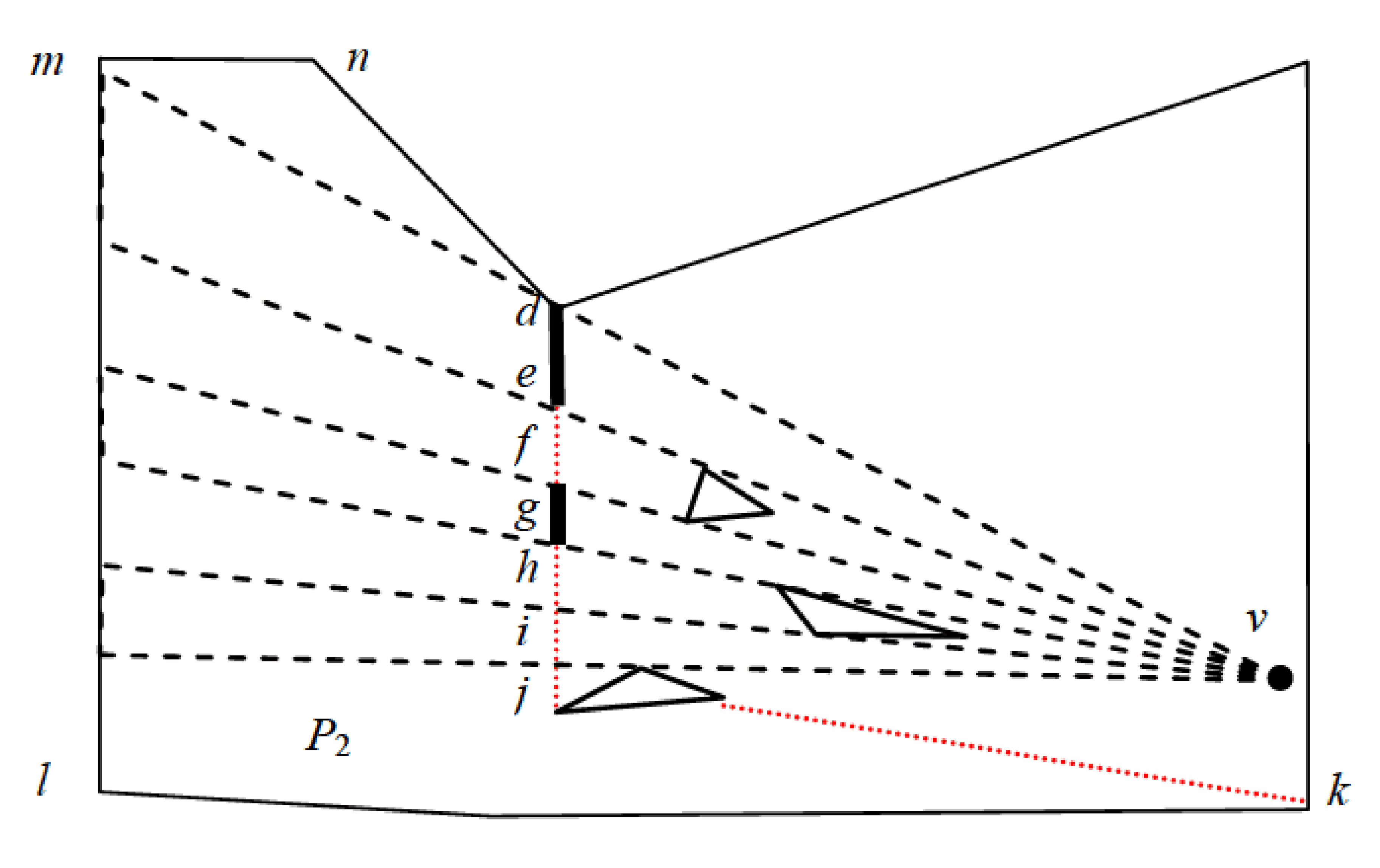


图 4.26　一个连接三角形边对应多个可见区域的情形

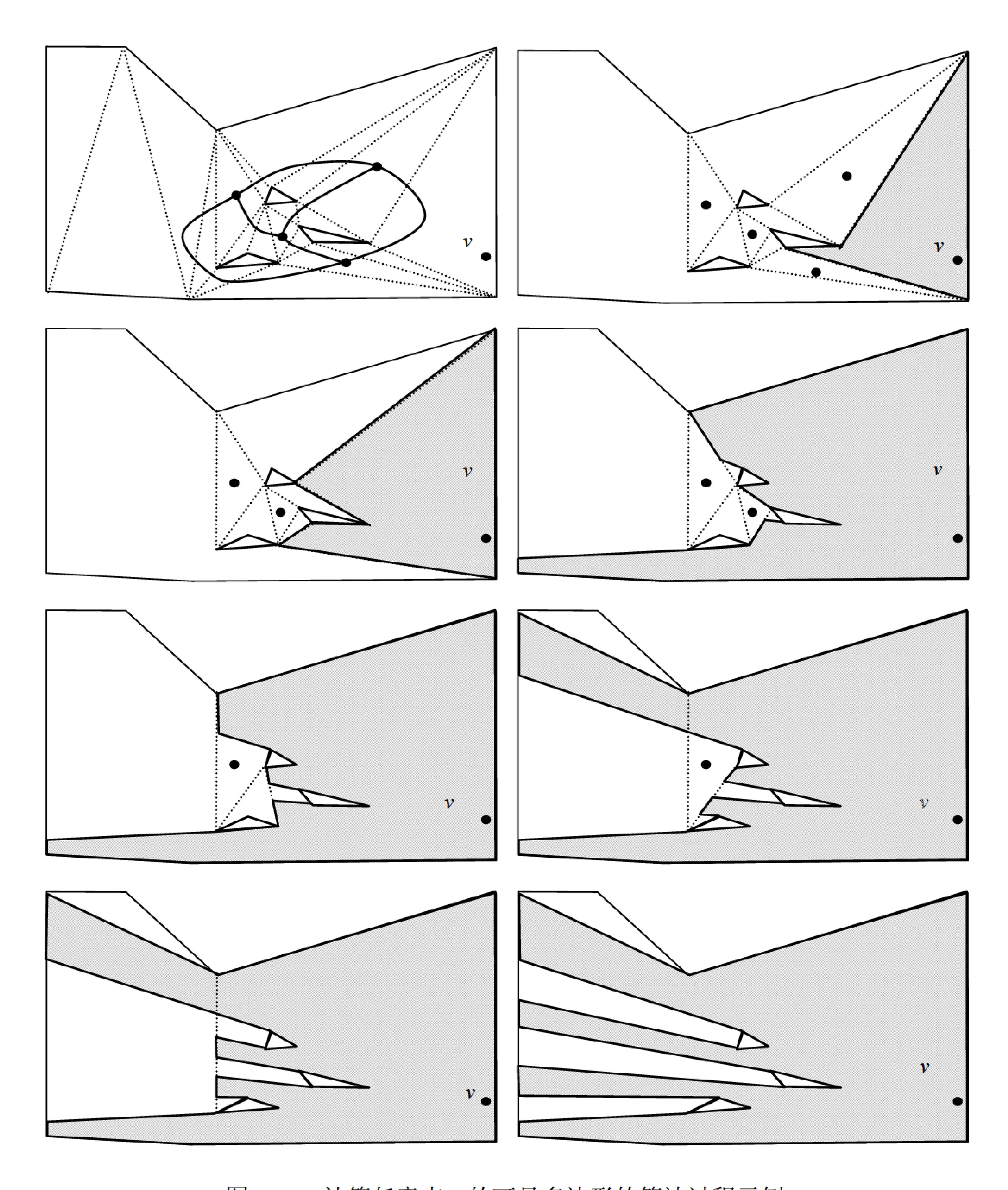


图 4.27　计算任意点 v 的可见多边形的算法过程示例

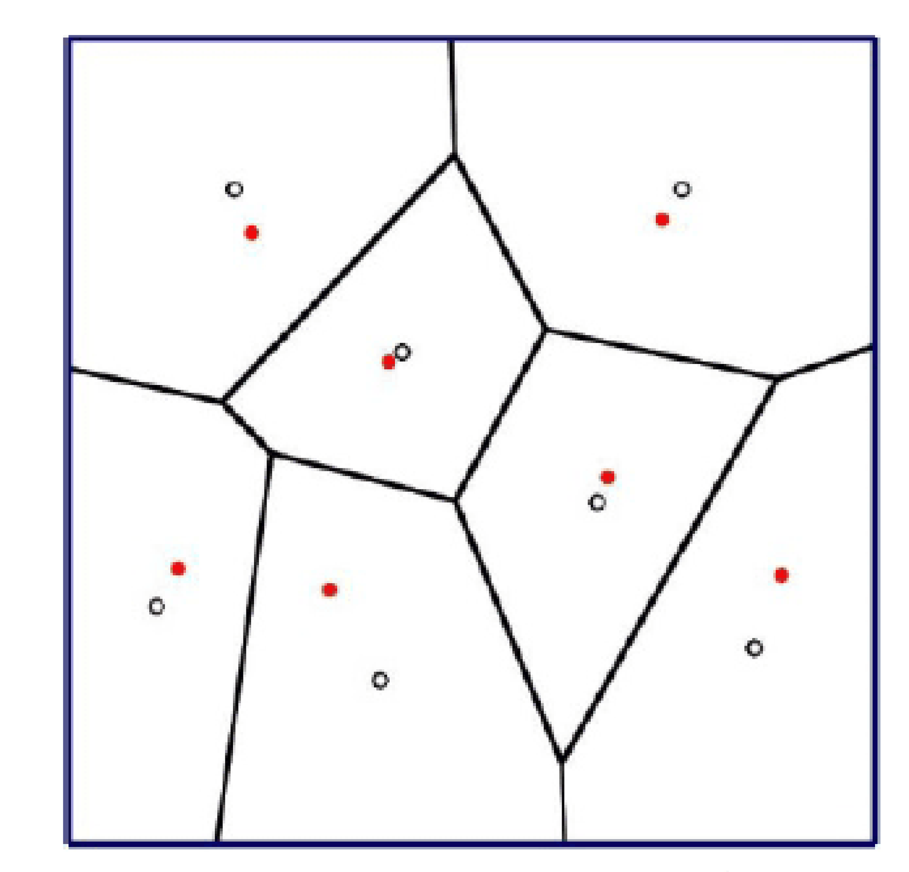
第 5 章　重心 Voronoi 图及其应用

5.1　定义与性质

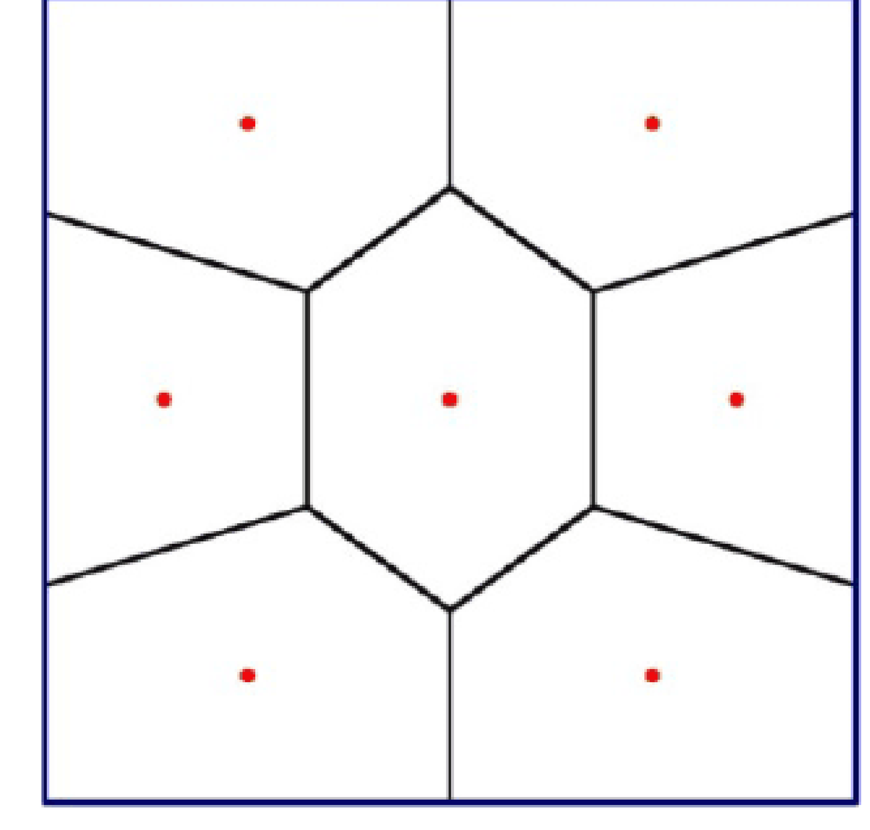
5.1.1　定义

如本书第 2 章所述，给定一个有限区域Ω和一组站点$X=(x_i)$，我们把每个站点x_i对应的 Voronoi 区域与Ω的交定义为x_i的单元，记做VR_i。一个单元VR_i的重心c_i定义为$c_i=\int_{VR_i} x\mathrm{d}\sigma / \int_{VR_i} \mathrm{d}\sigma$，其中$\mathrm{d}\sigma$为面积积分算子。如果每个站点$x_i$和它的单元$VR_i$的重心$c_i$重合，我们称$X$的 Voronoi 图为$\Omega$的重心 Voronoi 图(Centroidal Voronoi Diagram) [Du1999]，也称为 Centroidal Voronoi Tessellation，即 CVT。图 5.1（a）是一般的 Voronoi 图，图 5.1（b）是重心 Voronoi 图。可以看到，相对于一般 Voronoi 图，重心 Voronoi 图中的单元更加规整。需要注意的是，本章只考虑有限区域Ω的密度函数为常数的情形，当密度函数不为常数时，需要在公式中增加密度函数项。

重心 Voronoi 图是一组特殊分布的站点的 Voronoi 图。给定区域Ω，任意指定的一组站点X，它的 Voronoi 图一般来讲不是重心 Voronoi 图。



(a) 一般 Voronoi 图，站点（实心点）与重心（空心小圆）不重合



(b) 重心 Voronoi 图，站点与重心重合

图 5.1　重心 Voronoi 图与一般 Voronoi 图的比较

5.1.2 性质

重心 Voronoi 图的站点集合也可以被定义为下述优化问题的解：

$$F(X)=\sum_{i=1}^{n}f(x_i)\text{，其中 } f(x_i)=\int_{VR_i}\|x-x_i\|^2\,\mathrm{d}\sigma \qquad (5.1)$$

$F(X)$是站点集合定义在 $\mathbb{R}^N(N=dn)$上的能量函数（或称为目标函数），其中 d 为空间的维数，VR_i 是点 x_i 的 Voronoi 单元，$f(x_i)$ 亦被称为点 x_i 的能量函数。可以证明，使上述能量函数 $F(X)$取极小值的点集的 Voronoi 图一定构成一个重心 Voronoi 图[Du1999]。

CVT 目标函数 $F(X)$的梯度可通过以下公式计算：

$\dfrac{\partial F}{\partial x_i}=2m_i(p_i-c_i)$，其中，$m_i=\displaystyle\int_{VR(x_i)\cap\Omega}\mathrm{d}\sigma$，$c_i$ 是 x_i 的 Voronoi 区域的重心。

Gersho [Gersho1979] 提出猜想认为，在渐进意义下，一个最优的 CVT 中的各 Voronoi 单元具有相同的形状。Tóth[Toth2001]则给出了证明，即在渐进意义下，二维凸区域中的最优 CVT 中的各 Voronoi 单元形状为正六边形。在三维空间中，Gersho 猜想目前已有数值证明[Du2005]。这个性质有很多应用，比如，在采样理论中，可以假设 Ω 表示一个被近似的空间（如一幅图像的颜色空间），每个 x_i 对应于一个样本（颜色映射的元素），区域 VR_i 对应原始空间 Ω 中距离 x_i 最近的子集，极小化能量函数 $F(X)$可以保证每个样本 x_i 表示 Ω 中相等的信息量[Newman1982]。在图 5.2 中，被近似的空间 Ω 是一个正六面体，可以看出，重心 Voronoi 图划分非常规则。这些性质也都适用于特殊的几何处理，读者可以参阅文献[Du1999，Okabe2000，Liu2009，Du2010，WangJY2011]以了解更多关于重心 Voronoi 图的理论以及应用。

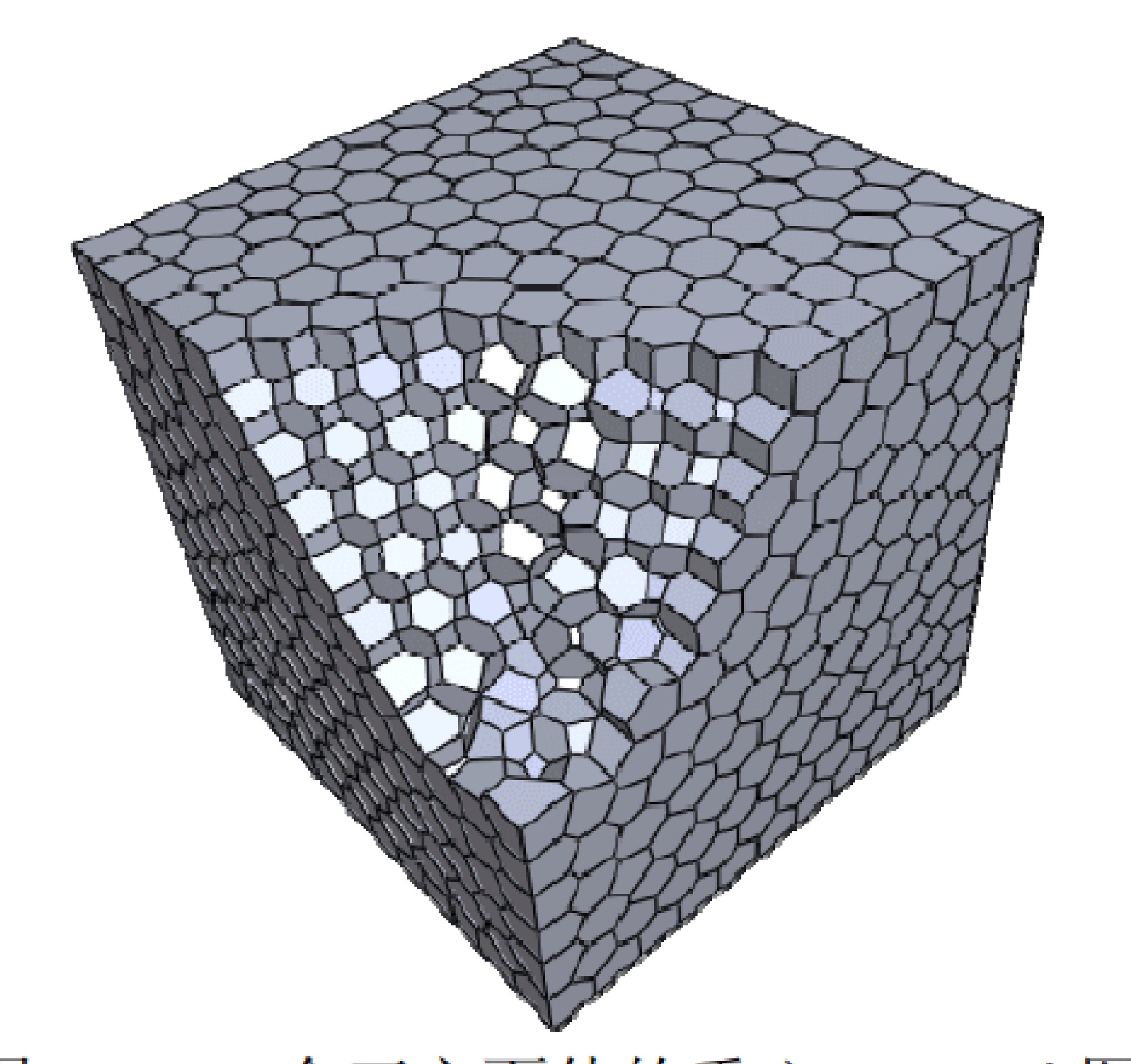


图 5.2　一个正六面体的重心 Voronoi 图

5.2　构 造 方 法

5.2.1　Lloyd 方法

Lloyd 方法[Lloyd1982]是计算给定区域 CVT 的基本算法，其通过一个收敛的迭代方法实现，属于确定性算法。具体步骤如下。

算法5.1　CVT-Lloyd算法

（1）初始化 n 个站点 $\{x_i\}_{i=1}^{n}$；

（2）对于每个点 $x_i(i=1\cdots n)$，计算与之对应的Voronoi区域 VR_i；

（3）对于每个Voronoi区域 $VR_i(i=1\cdots n)$，计算其重心，然后用重心更新站点 x_i；

（4）检测退出条件，如果满足则退出，否则返回第（2）步。

这个算法本质上讲是一个定点迭代的过程，迭代过程中减少聚类能量，从而可以保证收敛到 CVT。

5.2.2　MacQueen 方法

MacQueen方法[MacQueen1967]是计算给定区域CVT的另一个基本算法，

但其不同于Lloyd方法。MacQueen方法基于Monte Carlo方法提出，属于概率方法。具体步骤如下。

算法5.2　CVT-MacQueen算法

（1）初始化n个站点$\{x_i\}_{i=1}^n$，以及与每个点相对应的计数器j_i为1，$i=1\cdots n$；

（2）用Monte Carlo方法随机生成区域内的一个样本点ω；

（3）找到离ω最近的站点，假设为x_i，用如下方法更新x_i和其计数器j_i：

$$x_i \leftarrow \frac{j_i x_i + \omega}{j_i + 1}, \quad j_i \leftarrow j_i + 1$$

（4）检测退出条件，如果满足则退出，否则返回第（2）步。

MacQueen 方法虽然不需要计算 Voronoi 区域和重心，但用 MacQueen 方法产生的站点集$\{x_i\}_{i=1}^n$最终仍是收敛到 CVT 的站点集。不过 MacQueen 方法基于 Monte Carlo 方法，所以要大量撒点来得到更好的结果。研究者提出了很多改进的方法以加速其收敛，例如通过均匀撒点或者并行算法可以大大加速 MacQueen 方法的收敛速度。

上述 Lloyd 和 MacQueen 方法都是基于迭代的，算法的退出条件常常是依据具体的应用而改变。常用的终止条件有设置迭代次数，或判断站点与其 Voronoi 区域重心的距离是否小于阈值等。

5.2.3　牛顿法

在公式（5.1）中，$F(X)$是CVT的目标函数，该函数已经被证明在凸区域中具有C^2连续性[Liu2009]，因此可以采用二次优化算法（如牛顿法），基本步骤如下。

当$\|\nabla F(X)\| > \varepsilon$时，

（1）求解 d，以满足$\nabla^2 F(X)d = -\nabla F(X)$；

（2）寻找合适步长α，以满足$F(X+ad)$值下降；

（3） $X \leftarrow X + ad$ 。

其中 $\nabla F(X)$ 和 $\nabla^2 F(X)$ 分别表示目标函数 $F(X)$ 的梯度和 Hessian 矩阵。由于计算 Hessian 矩阵及其求逆相当耗时，在实际中可以采用 L-BFGS 算法(Limited-memory BFGS)[LBFGS]，即不计算二阶导数，而是通过利用以前的梯度和修改值去近似 Hessian 矩阵的逆矩阵。L-BFGS 算法具有近似的算法结构，不同之处在于在步骤（1）中 Hessian 矩阵被替换为由以前的梯度值构造的简单矩阵。

5.3 应用实例

5.3.1 基于无向图的重心 Voronoi 图的骨架匹配与模型分割

重心 Voronoi 图由于具有空间均匀采样的性质，有很多相关的应用和推广。本节介绍一种基于无向图的重心 Voronoi 图，及其应用于骨架匹配与模型分割的算法[Lu2012]。给定一个相互连接的直线段集合（即一个三维空间中的无向图）和一个三维几何模型，算法通过计算该无向图在几何模型中的重心 Voronoi 图，优化无向图的顶点位置，使得无向图的各条边能够最佳匹配所给模型的形状，并实现模型的合理分割。算法简单、自动，并且只需要一个参数（正则化权重），如图 5.3 所示。

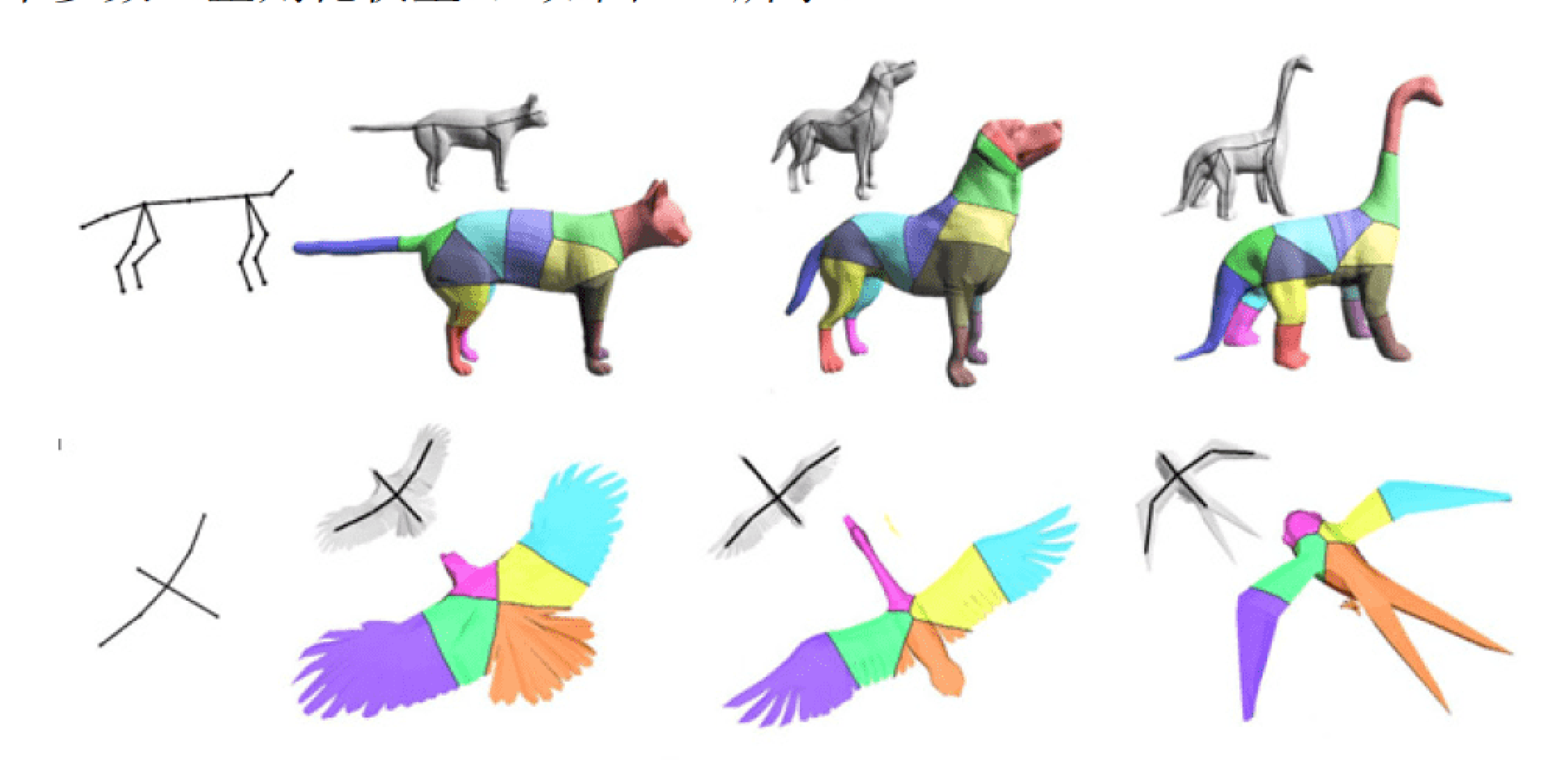


图 5.3　本方法处理四足动物和鸟的实验结果，左侧是原始骨架

1. 无向图的重心 Voronoi 图

给定一个有限区域Ω和一个无向图$G=(X,E)$，其中X是顶点集合，E是边的集合。这里定义G中任一边$[x_i,x_j]$的 CVT 能量值如下：

$$g([x_i,x_j])=\int_{VR([x_i,x_j])\cap\Omega}\mathrm{d}(z,[x_i,x_j])^2\mathrm{d}\sigma$$

其中，$VR([x_i,x_j])=\{z\in R^N\mid \mathrm{d}(z,[x_i,x_j])\leqslant \mathrm{d}(z,[x_k,x_1]),\forall k,1\in E\}$，$\mathrm{d}(z,[x_i,x_j])$定义了点到直线段的欧氏距离。

则无向图$G=(X,E)$的重心 Voronoi 图即使如下目标函数$F(X)$取得最小值的顶点集合X的分布：

$$F(X)=\sum_{i,j\in E}g([x_i,x_j]) \tag{5.2}$$

一般地，与点集的 Voronoi 图不同的是，在线段集的 Voronoi 图中，直接根据以上定义对函数$F(X)$进行优化求解是很困难的，主要有以下两点原因。

（1）线段 Voronoi 图的计算复杂性很高。两个线段的 Voronoi 中分线是二次曲线，生成线段的 Voronoi 图涉及二次曲线求交等计算，代价极高。

（2）即使得到了线段的 Voronoi 图，在其 Voronoi 单元边界上进行二次函数的积分也很困难。

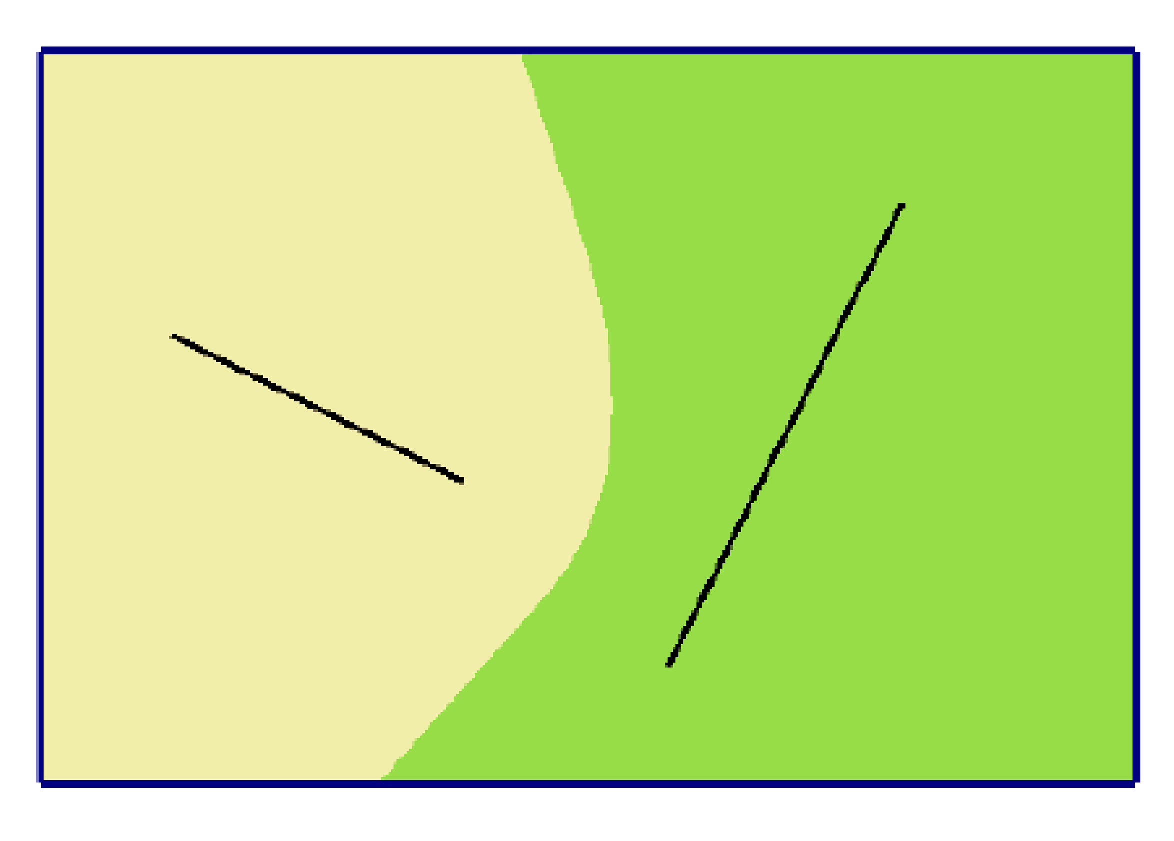
基于此两点原因，在计算中可采用近似的重心 Voronoi 图，即用一系列的采样点p_k代替直线段$[x_i,x_j]$：$p_k=\lambda_k x_i+(1-\lambda_k)x_j,\lambda_k\in[0,1]$，如图 5.4 所示。因此，线段的 Voronoi 图可以用所有采样点的 Voronoi 图的并集近似：

$$VR([x_i,x_j])\bullet\bigcup_k VR(p_k)$$

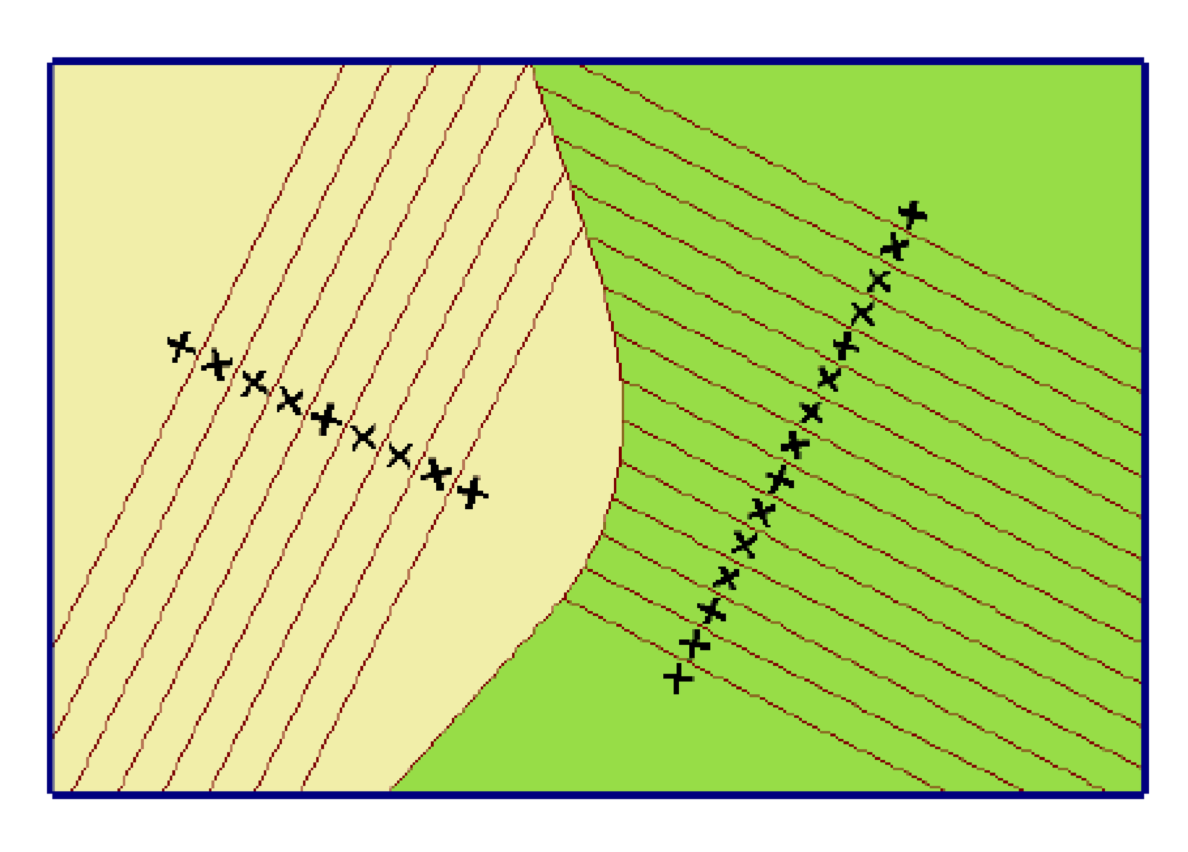
线段的能量值可以用所有采样点的能量之和近似：

$$g([x_i,x_j])\bullet\sum_k f(p_k)$$

其中，$f(p_k)=\int_{VR(p_k)}\|x-p_k\|^2\,\mathrm{d}\sigma$，表示点的 CVT 能量值，参见公式（5.1）。



(a) 精确 Voronoi 图



(b) 近似 Voronoi 图

图 5.4　两条线段 Voronoi 图的近似计算

基于以上讨论，$G=(X,E)$ 的 CVT 目标函数可近似表示为 $\tilde{F}(X)$，即 $G=(X,E)$ 的近似 CVT 是使目标函数 $\tilde{F}(X)$ 取得最小值的顶点 X 分布。目标函数 $\tilde{F}(X)$ 的定义如下：

$$\tilde{F}(X)=\sum_{i,j=E}\sum_{k}f(p_k)=\sum_{i,j=E}\sum_{k}f(\lambda_k x_i+(1-\lambda_k)x_j) \qquad (5.3)$$

因为近似目标函数 $\tilde{F}(X)$ 基于标准的点集 Voronoi 图定义，可用 Lloyd 能量计算，它比初始目标函数（见公式（5.2））更容易优化。注意，近似目标函数与初始目标函数依赖于相同的变量 X。线段的采样点 $p_k=\lambda_k x_i+(1-\lambda_k)x_j$ 不是变量，因为它们仅仅依赖于两个端点 x_i 和 x_j。

为了避免产生退化的极小值，像在变分曲面设计中的做法一样，引入了一个正则项 $R(X)$ [Mal1989]，以避免不必要的振荡，调整无向图和给定形状以更好的拟合。该正则项采用拉普拉斯能量处理无向图中度数大于 1 的顶点，定义如下：

$$R(X)=\sum_{x_i,V(x_i)>1}\left\|x_i-\frac{1}{V(x_i)}\sum_{x_j\in N(x_i)}x_j\right\|^2 \qquad (5.4)$$

其中，$V(x_i)$ 表示顶点的度数，$N(x_i)$ 表示 x_i 的相邻顶点。

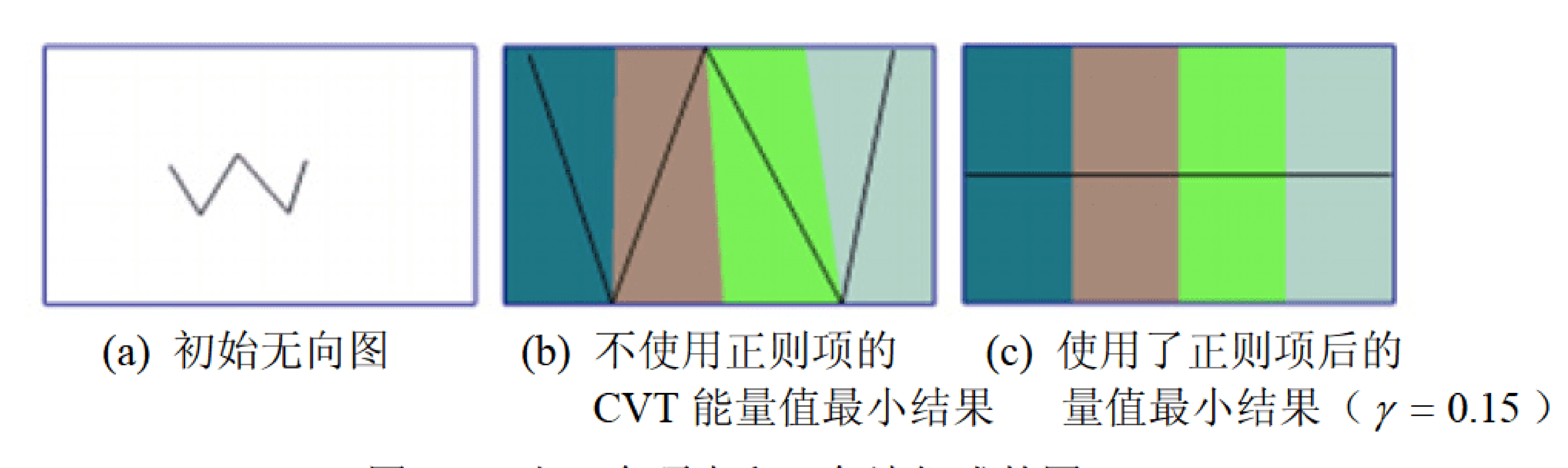
综合利用公式（5.3）和公式（5.4），定义用于三维形状骨架匹配的目标

函数为：

$$H(X)=\tilde{F}(X)+\gamma|\Omega|R(X) \tag{5.5}$$

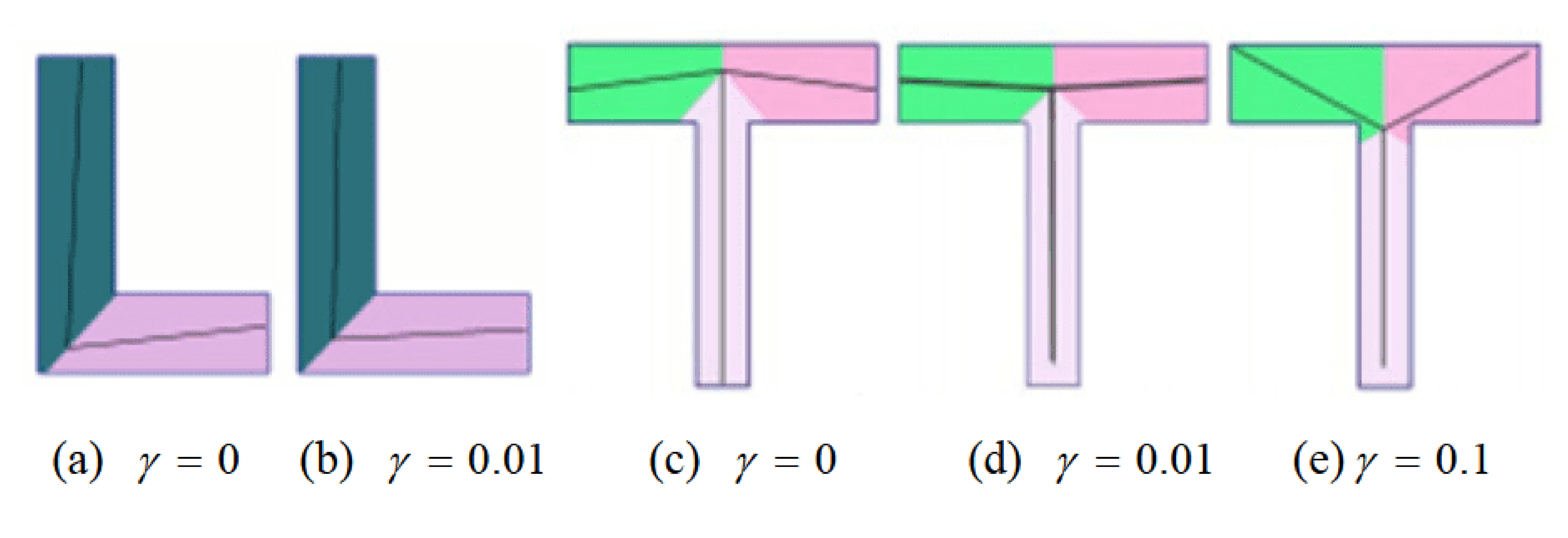
其中，$\tilde{F}(X)$ 是线段的近似 Lloyd 能量值，$R(X)$ 是正则项，$\gamma \in R^{+}$ 表示正则项的影响。

为了进一步说明正则项的作用，参见图 5.5。一般情况下，极小化 CVT 目标函数趋向于使 Voronoi 区域的紧凑程度最大化，但是在一些特殊情况下可能会导致振荡误差，对于骨架匹配这类应用亦不适用。如在图 5.5 中，图 5.5（b）比图 5.5（c）有更小的 CVT 能量值，但图 5.5（c）是考虑了正则项之后的结果，作为骨架更好地匹配了给定形状。



(a) 初始无向图　(b) 不使用正则项的 CVT 能量值最小结果　(c) 使用了正则项后的量值最小结果（$\gamma=0.15$）

图 5.5　由 5 个顶点和 4 条边组成的图



(a) $\gamma=0$　(b) $\gamma=0.01$　(c) $\gamma=0$　(d) $\gamma=0.01$　(e) $\gamma=0.1$

图 5.6　正则化参数 γ 的影响

图 5.6 展示了参数 γ 对结果的影响。对于度为 2 的结点，γ 越大则与其相关联的两条线段的夹角（小于 π）越大。对于度大于 2 的结点，γ 越大，则与其相关联的线段的夹角以及线段的长度越均匀。实验表明，二维空间中 γ 取 0.01，三维空间中 γ 取 0.02，能够得到较好的结果。

2. 解决机制

（1）产生采样点

首先选取一个采样步长 h，然后沿着直线段每隔步长 h 取一个采样点。线段的端点（度为 1）也加入到采样点集合中。度大于 1 的顶点不加入采样点，这是为了在分支点处获得对平分线比较好的近似。另外一种采样方法是用固定数目的点分割线段。实验结果表明，这两种方法得到的结果差异很小。

（2）计算裁剪 Voronoi 区域

算法先计算出采样点集 p_k 的 Delaunay 三角剖分，并通过计算 Delaunay 三角剖分的对偶图得到 Voronoi 图。然后计算出每个 Voronoi 区域 $VR(p_k)$ 和区域 Ω（一个三角化曲面的内部）的交线。因为 Voronoi 区域是凸的，所以容易用 Voronoi 区域去裁剪三角化曲面。为此，算法采用了经典的 Sutherland-Hodgman 逐边裁剪算法[Sutherland1974]。

（3）计算能量和梯度值

基于 CVT 的变分目标函数定义，算法采用 L-BFGS 拟牛顿法进行优化求解，在每次迭代时，这种方法需要计算目标函数的能量值以及梯度。

首先考虑 Lloyd 能量 $\tilde{F}(X)$ 的近似值。$\tilde{F}(X)$ 可以被分解为一些项的和，每个项都在裁剪过程中对应一个四面体。与一个四面体 $T=(p_k,q_1,q_2,q_3)$ 相关的 CVT 能量是：

$$\frac{|T|}{10}(U_1^2+U_2^2+U_3^2+U_1\cdot U_2+U_2\cdot U_3+U_3\cdot U_1)$$

其中 $U_i=q_i-p_k$，$|T|$ 表示四面体的有向体积。为了计算梯度，可以参考 5.1.2 节，通过代入表达式

$p_k=\lambda_k x_i+(1-\lambda_k)x_j$，得到边 $[x_i,x_j]$ 的 CVT 能量：

$$\frac{\partial\tilde{F}}{\partial x_i}=\sum_k\frac{\partial f(p_k)}{\partial x_i}=\sum_k\frac{\partial f(p_k)}{\partial p_k}\frac{\partial p_k}{\partial x_i}=2\sum_k m_k(p_k-c_k)\lambda_k$$

$$\frac{\partial \tilde{F}}{\partial x_j}=\sum_k \frac{\partial f(p_k)}{\partial x_j}=2\sum_k m_k(p_k-c_k)(1-\lambda_k)$$

m_k 和 c_k 分别表示 p_k 所在 Voronoi 区域的质量和重心。m_k 和 c_k 可以通过裁剪 Voronoi 区域计算出来，裁剪 Voronoi 区域用一些有向四面体的并集表示。

正则项 $R(X)$ 对梯度的贡献是：

$$\frac{\partial R}{\partial x_i}=2\left(x_i-\frac{1}{V(x_i)}\sum_{x_j\in N(x_i)} x_j\right)-2\sum_{x_j\in N(x_i),V(x_j)>1}\frac{1}{V(x_j)}\left(x_j-\frac{1}{V(x_j)}\sum_{x_k\in N(x_j)} x_k\right)$$

根据以上步骤，可求出目标函数 $H(X)$ 的能量与梯度，采用 L-BFGS 拟牛顿法优化求解，即可获得给定三维形状的良好骨架匹配以及相应的表面分割，如图 5.3 所示。

5.3.2　基于流线重心 Voronoi 图的流场可视化

流场可视化是科学可视化中非常重要的组成部分，特别在计算力学、流体力学和空气动力学、天气预报和地球物理模拟等领域有着广阔的应用前景。归纳来说，流场的可视化技术可以分为五类[Brambilla2012]：直接可视化、基于纹理的可视化、基于特征的可视化、基于几何的可视化和基于空间剖分的可视化。基于几何的方法采用流线（Streamline）、流面（Stream Surface）、流管（Stream Tube）、迹线（Pathline）、脉线（Streakline）、时线（Timeline）等几何元素来表示流场，具有表示清晰、简洁等优点。

流线、迹线等基于粒子的方法是基于几何的可视化方法中最常用的技术，流线即是处处与流场相切的曲线，能有效地表现流场的局部特征。对于一条流线而言，可以通过自种子点开始进行数值积分的方法获得，因此如何选取种子点的位置从而均匀布置流线是此类方法的关键所在。本节将离散点的重心 Voronoi 图的概念推广到流线上，提出流线重心 Voronoi 图的概念，并给出一种新的流线布置方法，可实现更佳效果的流场可视化[Liu2013]。本算法是面向二维定常流场进行介绍的，但也可推广至非定常流场。

1. 流线的重心 Voronoi 图

流线是某一时刻在流场中画出的一条空间曲线，在该时刻，曲线上所有

质点的速度矢量均与这条曲线相切。流线是基于欧拉方法描述流场的一种手段，在实际应用中常用多线段（polyline）的形式来表示，即由种子点出发，由数值积分的方法跟踪得到的点序列。但是如何选择积分的步数并不是一件简单的事情。由于长流线可以更好地表现出流线的连续性，本节中的算法选择使用长流线作为表现流场的几何元素。当算法中的长流线遇到流场边界或速度为 0 的点时，停止积分。我们用 s_i 表示流线，用 $S=(s_i)_{i=1}^n$ 表示给定流场的一组流线，将每条流线 s_i 作为站点计算重心 Voronoi 图，在区域 Ω 中其 CVT 能量可以定义为：

$$g([s_i])=\int_{VR([s_i])\cap\Omega} d(z,s_i)^2 \mathrm{d}\sigma \tag{5.6}$$

其中，$VR([s_i])=\{z\in R^N \mid d(z,[s_i])\leqslant d(z,[s_j]),\forall j\neq i\in S\}$，$d(z,[s_i])$ 表示点 z 到流线 s_i 的欧式距离。

定义 5.1　流线集合 $S=(s_i)_{i=1}^n$ 的重心 Voronoi 分布是目标函数

$$G(S)=\sum_{i\in S} g([s_i])$$

的一个极小解，其中 $g([s_i])$ 见公式（5.6）。

类似 5.3.1 节所属，直接根据以上定义对函数 $G(S)$进行优化求解是很困难的。为此，算法采用近似计算的方式计算流线的 CVT 函数值。我们用流线上的点 $\{p_1,\ldots,p_m\}$ 来代替每条流线 $s[x_i]$。这里 x_i 是 $s[x_i]$ 的种子点，用 $p_j=x_i$ 来表示它，$p_1,\ldots,p_m$ 是由欧拉积分法或龙格-库塔积分法等方法进行流线追踪得到的点序列，在下文中也称为采样点。因此，流线 $s[x_i]$ 的 Voronoi 单元可以用生成该流线的所有积分点序列的 Voronoi 单元的并集近似表示，即：

$$VR(s[x_i])\cong\bigcup_k VR(p_k)$$

因此，流线的能量值可以用所有采样点的能量之和近似：

$$g(s[x_i])\cong\sum_k f(p_k)$$

其中，$f(p_k)$表示点 p_k 的 CVT 能量（见公式 5.1）。下文中也用所有种子点

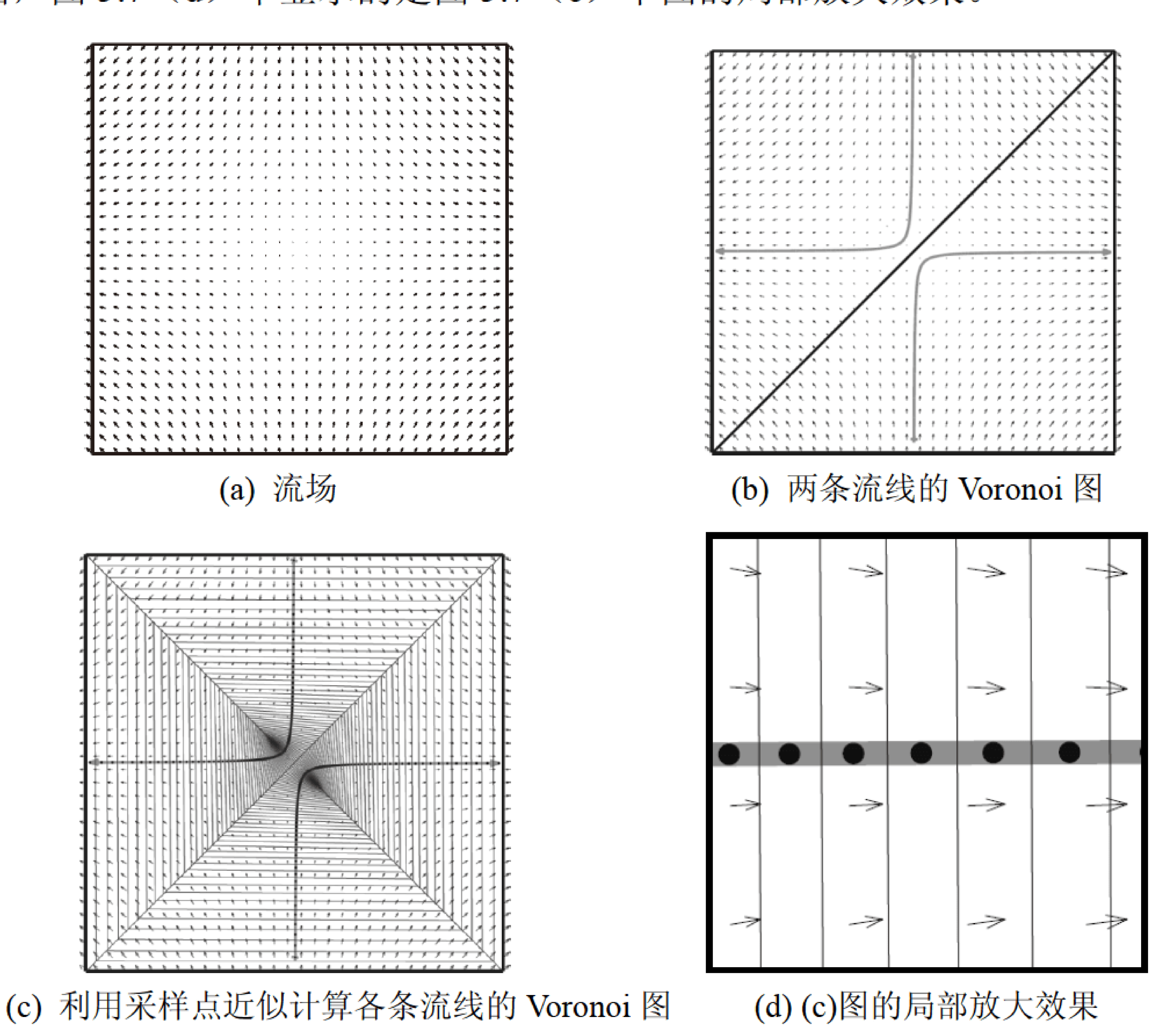
的集合 $X=(x_i)_{i=1}^{n}$ 来表示流线集合 S，因此 S 的 Voronoi 图可以由所有流线生成点的 Voronoi 图近似表示。

定义 5.2 流线 S 的近似重心 Voronoi 分布是目标函数

$$\tilde{F}(X)=\sum_{i\in S}\sum_{k}f(p_k) \tag{5.7}$$

的一个极小解。

图 5.7 给出了一个流线 Voronoi 图的近似计算示例。其中，对于图 5.7（a）中的流场，图 5.7（b）中显示的是两条流线（曲线）的 Voronoi 图（中间直线段），图 5.7（c）中给出了利用采样点近似计算得到的各条流线的 Voronoi 图，图 5.7（d）中显示的是图 5.7（c）中图的局部放大效果。



(a) 流场　(b) 两条流线的 Voronoi 图

(c) 利用采样点近似计算各条流线的 Voronoi 图　(d) (c)图的局部放大效果

图 5.7　流线 Voronoi 图的近似计算示例

2. 流场可视化算法

基于流线重心 Voronoi 图的定义，可知其优化目标是利用流线对于给定流场进行均匀采样。算法就是利用这个原理优化流线的布置。给定一个流场和一定数目的流线，通过极小化流线的 CVT 能量，从而达到优化流线布置，直至在流场中均匀分布。

为了极小化目标函数 $\tilde{F}(X)$ （见公式 5.7），算法利用拟牛顿法求解。这里的目标函数 $\tilde{F}(X)$ 是点 CVT 函数的线性组合，因此同样具有 C^2 连续性，即可采用牛顿法或拟牛顿法优化求解。在实践中可以采用优化工具包 TAO[Munson2012]或科学计算工具包 PETSc[Balay2011]等提供的 L-BFGS 算法实现，算法框架如下。

对于每个牛顿迭代，有：

（1）对于每条流线 $s[x_i]$，计算其采样点 $\{p_k\}_{k=1}^{m}$；

（2）对于每个采样点 p_k，计算其裁剪 Voronoi 区域，即 $VR(p_k)\cap\Omega$，其中 Ω 表示给定区域；

（3）将每个采样点 p_k 的贡献加到 $\tilde{F}(X)$ 和 $\nabla\tilde{F}(X)$ 中。

（1）生成采样点

给定种子点 x_i，可以通过欧拉积分或龙格-库塔积分计算其流线上的采样点。欧拉积分法可以表示为：$p_{k+1}=p_k+h\cdot V(p_k)$，其中 $p_j=x_i$，$k=j\cdots m$，h 是积分步长，流线可以通过同时向前向后积分获得。二次龙格-库塔积分法可以表示为：$p'_k=p_k+\frac{1}{2}h\cdot V(p_k), p_{k+1}=p_k+h\cdot V(p'_k)$ [Press2002]。在算法实现中，为了方便计算梯度，将流线的起始点 x_i 以及其流场方向 V_i 和 V_i 正交向量 $V_i^{\perp}$ 作为参考标架，将采样点 p_k 表示为该标架中的向量，即 $p_k=x_i+\alpha_k V_i+\beta_k V_i^{\perp}$。

得到流线上的采样点之后，可以直接计算每个采样点在给定区域边界内的 Voronoi 单元，从而计算 $\tilde{F}(X)$。

（2）计算 $\tilde{F}(X)$ 和 $\nabla\tilde{F}(X)$

对于一个三角形△ $p_k q_1 q_2$，它所关联的重心 Voronoi 函数能量可以通过下式计算得到：$\frac{|\triangle|}{6}(U_1^2+U_2^2+U_1\cdot U_2)$，其中 $U_i=q_i-p_k$，$|\Delta|$ 表示三角形的带符号面积。重心 Voronoi 函数的梯度可以通过下式计算得到：$\frac{\partial f(p_k)}{\partial p_k}=2m_k(p_k-c_k)$，其中，$m_k$ 和 c_k 分别表示三角形的质量和重心。$\tilde{F}(X)$ 可以通过下式计算：

$$\frac{\partial \tilde{F}}{\partial x_i}=\sum_k \frac{\partial f(p_k)}{\partial x_i}=\sum_k \frac{\partial f(p_k)}{\partial v_k}\frac{\partial p_k}{\partial x_i}=2\sum_k m_k \frac{\partial p_k}{\partial x_i}(p_k-c_k)$$

流线上的采样点相对于种子点的梯度如下：

$$\frac{\partial p_k}{\partial x_i}=I+\alpha_k\begin{bmatrix}\frac{\partial V_{ix}}{\partial x} & \frac{\partial V_{iy}}{\partial x}\\ \frac{\partial V_{ix}}{\partial y} & \frac{\partial V_{iy}}{\partial y}\end{bmatrix}+\beta_k\begin{bmatrix}\frac{-\partial V_{iy}}{\partial x} & \frac{\partial V_{ix}}{\partial x}\\ \frac{-\partial V_{iy}}{\partial y} & \frac{\partial V_{ix}}{\partial y}\end{bmatrix}$$

（3）实验结果

算法基于计算几何算法库 CGAL[CGAL]，用 C/C++语言进行了实现。算法将流场区域归一化为边长为 1 的正方形，积分步长 h 取值为 0.01。在实现中，当流线遇到边界或达到积分步数时停止积分。实验结果表明，本算法的结果比 CGAL 中的算法更简洁、规整，同时具有更大的灵活性，也允许用户交互。图 5.8 给出了本方法对于四个不同流场的可视化结果。

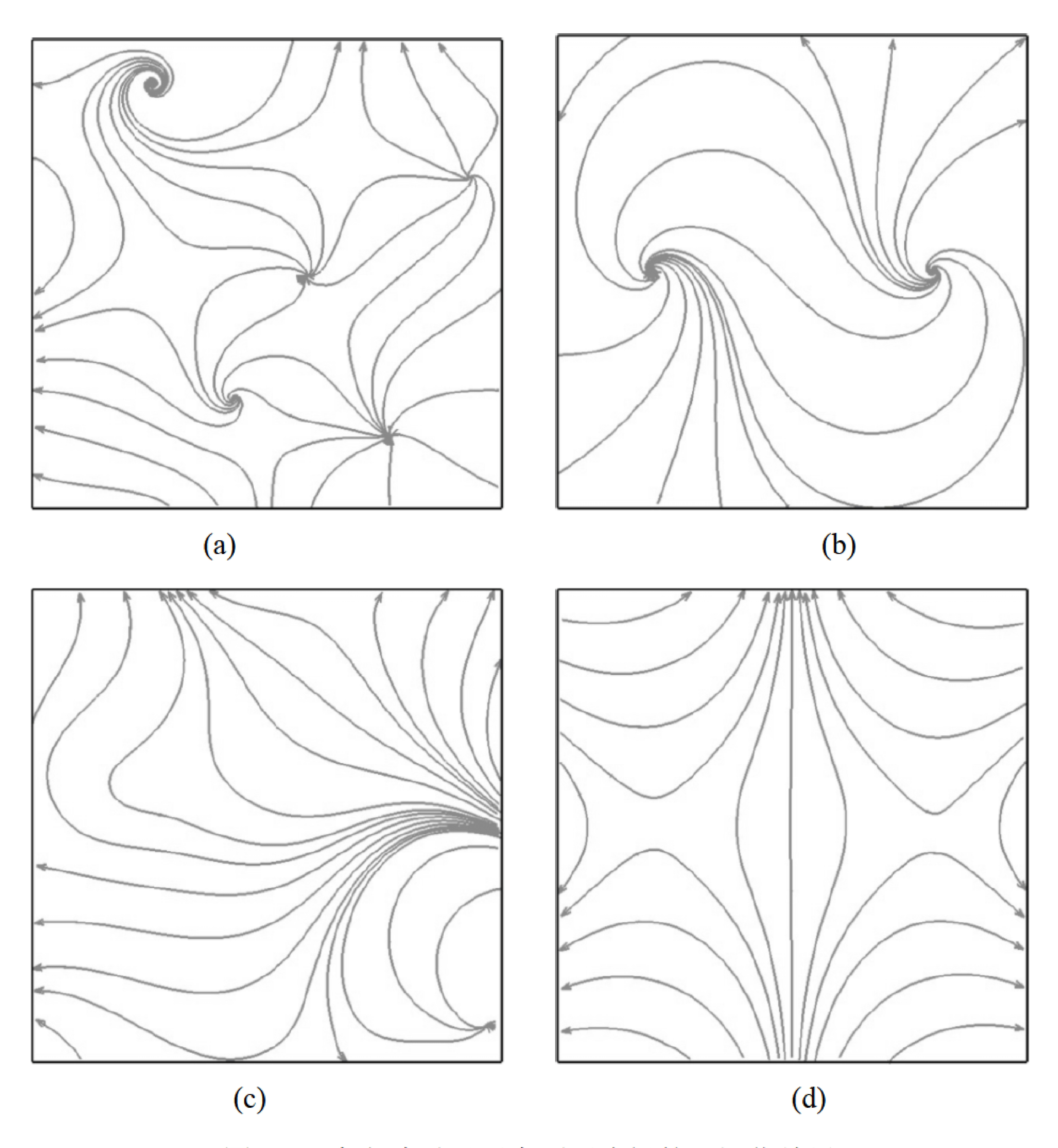


图 5.8　本方法对于四个不同流场的可视化结果

参 考 文 献

[Aggarwal1989] Alok Aggarwal, Leonidas J. Guibas, James Saxe, Peter W. Shor. A Linear-Time Algorithm for Computing the Voronoi Diagram of a Convex Polygon. Discrete and Computational Geometry. 1989, 4(1):591-604

[Aronov2002] Boris Aronov, Leonidas J. Guibas, Marek Teichmann, Li Zhang. Visibility Queries and Maintenance in Simple Polygons. Discrete and Computational Geometry. 2002, 27(4):461–483

[Aurenhammer1991 Franz Aurenhammer.Voronoi diagrams—a survey of a fundamental geometric data structure, ACM Computing Surveys. 1991, 23(3): 345-405

[Balay2011] Satish Balay, Jed Brown, Kris Buschelman, Victor Eijkhout, William D. Gropp, Dinesh Kaushik, Matthew G Knepley, Lois Curfman McInnes, Barry F. Smith, and Hong Zhang. PETSc users manual. Technical Report ANL-95/11 - Revision 3.2. Argonne National Laboratory. 2011

[Berg2008] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. Computational Geometry: Algorithms and Applications (3rd edition) Springer-Verlag TELOS Santa Clara, CA, USA , 2008

[Blum1967] H. Blum. A transformation for extracting new description of shape. In: Wathen-Dunn. Wed., Model for the perception of Speech and Visual Form. Cambride. Massachusetts: MIT Press.1967, pp.362-380

[Boissonnat1992] Jean-Daniel Boissonnat, Olivier Devillers, René Schott, Monique Teillaud, Mariette Yvinec. Applications of random sampling to on-line algorithms in computational geometry. Discrete and Computational Geometry. 1992, 8(1): 51-71.

[Brambilla2012] Andrea Brambilla, Robert Carnecky, Ronald Peikert, Ivan Viola, and Helwig Hauser. Illustrative flow visualization: State of the art, trends and challenges. Computer Graphics Forum. 2012, 31(2):75-94

[Cai2010] 蔡强. 限定 Voronoi 网格剖分的理论及应用研究. 北京：北京邮电大学出版社，2010

[CGAL]CGAL. Computational Geometry Algorithms Library. http://www.cgal.org

[Chazelle1982] B. Chazelle, Bernard Chazelle, Bernard Chazelle, Bernard Chazelle. A Theorem on Polygon Cutting with Applications. In : Proceeding of 23rd Annual Symposium on Foundations of Computer Science. 1982, pp. 339-349

[Chen2004] 陈军. Voronoi 动态空间数据模型. 北京：测绘出版社，2004

[Chin1995] Francis Chin, Jack Snoeyink, Cao An Wang. Finding the Medial Axis of a Simple Polygon in Linear Time. In: LNCS1004, 1995, pp. 382-391(Proceeding of 6th Annual International Symposium on Algorithms and Computations)

[Chin1998] Francis Yuk-Lun Chin and Cao An Wang. Finding the constrained Delaunay Triangulation and constrainedVoronoi Diagram of a Simple Polygon in linear time. SIAM Journal on Computing.1998, 28(2):471-486

[Denny2003] Markus Oswald Denny. Algorithmic Geometry via Graphics Hardware. PhD dissertation, Universitat des Saarlandes. 2003

[Descartes1644] Rene Descartes. Principia Philosophiae. Amsterdam: Ludovicus Elzevirius. 1644

[Dirichlet1850] G Lejeune Dirichlet. Über die Reduction der positiven quadratischen Formen mit drei unbestimmten ganzen Zahlen. Journal für die Reine und Angewandte Mathematik. 1850, 40: 209-227

[Du1999] Qiang Du, Vance Faber, and Max Gunzburger. Centroidal Voronoi Tessellations: Applications and Algorithms. Society for Industrial and Applied Mathematics. 1999, 41(4): 637-676

[Du2005] Qiang Du, Desheng Wang. The Optimal Centroidal Voronoi Tessellations and

the Gersho's Conjecture in the Three-Dimensional Space. Computers and Mathematics with Applications, 2005, 49(9/10): 1355-1373

[Du2010] Qiang Du, Max Gunzburger, Lili Ju. Advances in Studies and Applications of Centroidal. Voronoi Tessellations. Numerical Mathematics: Theory, Methods and Applications. 2010, 3(2): 119-142

[Edelsbrunner1990] Herbert Edelsbrunner, Ernst Peter Mucke. Simulation of Simplicity: A Technique to Cope with Degenerate Cases in Geometric Algorithms. ACM Transactions on Graphics. 1990,9(1):66-104

[Edelsbrunner2001] Herbert Edelsbrunner. Geometry and Topology for Mesh Generation. New York: Cambridge University Press. 2001

[Fan2010] 樊广佺. 计算几何若干方法及其在空间数据挖掘中的应用. 北京：冶金工业出版社，2010

[Fang1993] Tsung Pao Fang, Les A. Piegl. Delaunay Triangulation Using a Uniform Grid IEEE Computer Graphics and Applications. 1993, 13(3): 36-47

[Fortune1987] Steven Fortune. A sweepline algorithm for Voronoi diagrams. Algorithmica. 1987, 2(1-4): 153-174

[Gersho1979] A. Gersho. Asymptotically Optimal Block Quantization. IEEE Transactions on Information Theory. 1979, 25(4). 373-380

[Goodman2004] Jacob E. Goodman and Joseph O'Rourke. Handbook of Discrete and Computational Geometry (2nd edition). CRC Press, Inc., Boca Raton, FL, USA, 2004

[Gold2006] http://www.voronoi.com

[Green1978] P. J. Green and R. Sibson. Computing Dirichlet Tessellations in the Plane, Computer Journal.1978,21(2):168-173

[Guibas1987] Leonidas J. Guibas, John Hershberger, Daniel Leven, Micha Sharir, Robert E. Tarjan. Linear-Time Algorithms for Visibility and Shortest Path Problems Inside Triangulated Simple Polygons. Algorithmica. 1987, 2: 209-233

[Guibas1988] Leonidas J. Guibas and Jorge Stolfi. Ruler, Compass and Computer. In RaeA.

Earnshaw, editor, Theoretical Foundations of Computer Graphics and CAD, volume 40 of NATO ASI Series, pages 111–165. Springer Berlin Heidelberg, 1988

[Held1991] Martin Held. On the Computational Geometry of Pocket Machining, volume 500 of Lecture Notes in Computer Science. Springer, 1991

[Held2001] Martin Held. VRONI: An Engineering Approach to the Reliable and Efficient Computation of Voronoi Diagrams of Points and Line Segments. Computational Geometry: Theory and Application. 2001, 18(2): 95-123

[Hjelle2006] Øyvind Hjelle and Morten Dæhlen. Triangulations and Applications (Mathematics and Visualization). Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006

[Hoff1999] Kenneth E. Hoff III, John Keyser, Ming Lin, Dinesh Manocha, Tim Culver. Fast Computation of Generalized Voronoi Diagrams Using Graphics Hardware. In: Proceedings of the 26th annual conference on Computer graphics and interactive techniques. 1999, pp.277-286

[Jaeger2000] S. Jaeger, S. Manke, J. Reichert, A. Waibel. Online Handwriting Recognition: The NPen++ Recognizer. International Journal on Document Analysis and Recognition. 2001, 3(3): 169-180

[Kenneth1996] Kenneth R. Castleman. Digital Image Processing. Prentice-Hall signal processing series. Prentice Hall, 1996

[Kirkpatrick1979] D. G. Kirkpatrick. Efficient Computation of Continuous Skeletons. In: Proceeding of 20th Annual Symposium on Foundations of Computer Science. 1979, pp.18-27

[Kirkpatrick1983] D. G. Kirkpatrick. Optimal Search in Planar Subdivisions. SIAM Journal on Computing. 1983, 12 : 28-35

[Klein1993] Rolf Klein, Kurt Mehlhorn, Stefan Meiser. Randomized Incremental Construction of Abstract Voronoi Diagrams. Computational Geometry. 1993, 3(3):157-184

[LBFGS] http://en.wikipedia.org/wiki/Limited-memory_BFGS

[Lee1982] D.T. Lee. Medial Axis Transformation of a Planar Shape. IEEE Transactions on Pattern Analysis and Machine Intelligence. 1982, 4(4): 363-369

[Li2010] 李海生. Delaunay 三角剖分理论及可视化应用研究. 哈尔滨: 哈尔滨工业大学出版社，2010

[Lin1991] Ming C. Lin, John F. Canny. A fast algorithm for incremental distance calculation. In: Proceeding of the IEEE International Conference on Robotics and Automation. 1991, pp.1008 -1014

[Lloyd1982] Stuart Lloyd. Least Square Quantization in PCM. IEEE Transactions on Information Theory. 1982, 28: 129-137

[Liu2009] Yang Liu, Wenping Wang, Bruno Levy, Feng Sun, Dong-Ming Yan, Lin Lu, and Chenglei Yang. On Centroidal Voronoi Tessellation—Energy Smoothness and Fast Computation. ACM Transactions on Graphics. 2009, 28(4):1-17

[Liu2013] Wenjie Liu, Lin Lu, Bruno Lévy, Chenglei Yang and Xiangxu Meng. Centroidal Voronoi Tessellation of Streamlines for Flow Visualization. The 10th International Symposium on Voronoi Diagrams in Science and Engineering, July 8-10, 2013, pp.75-81

[Lu2011] Lin Lu , Chenglei Yang, and Jiaye Wang. Point Visibility Computing in Polygons with Holes. Journal of Information and Computational Science. 2011, 8(16) :4165-4173

[Lu2012] Lin Lu, Bruno Lévy, and Wenping Wang. Centroidal Voronoi Tessellation of Line Segments and Graphs. Computer Graphics Forum(Eurographics 2012). 2012, 31(2) : 775-784

[MacQueen1967] James MacQueen. Some methods for classification and analysis of multivariate observations. In : Proceeding of Fifth Berkeley Symposium on Mathematical Statistics and Probability, Vol I. 1967, pp.281-297

[Mal1989] Jean-Laurent Mallet. Discrete smooth interpolation. ACM Transactions on Graphics. 1989, 8 (2): 121-144

[Mebarki2005] Abdelkrim Mebarki, Pierre Alliez, Olivier Devillers. Farthest point seeding for efficient placement of streamlines. In: IEEE Visualization 2005. 2005, pp. 479-486

[Meng2010] 孟雷，张俊伟，王筱婷，杨承磊. 一种改进的 Voronoi 图增量构造算法. 中国图象图形学报. 2010，15(06): 978-984

[Munson2012] Todd Munson, Jason Sarich, Stefan Wild, Steven Benson, and Lois Curfman McInnes. Tao 2.0 users manual. Technical Report ANL/MCS-TM-322. Mathematics and Computer Science Division. Argonne National Laboratory. 2012. http://www.mcs.anl.gov/tao

[Newman1982] Donald Newman. The Hexagon theorem. IEEE Transactions on Information Theory. 1982, 28(2):137-139

[Okabe2000] Atsuyuki Okabe, Barry Boots, Kokichi Sugihara, Sung Nok Chiu. Spatial Tessellations: Concepts and Applications of Voronoi Diagrams (2nd edition). Chichester: Wiley. 2000

[ORourke1998] Joseph O'Rourke. Computational Geometry in C (2nd edition). Cambridge University Press, New York, NY, USA, 1998

[Peter2002] Peter H. Chou, Teresa HY Meng. Vertex Data Compression through Vector Quantization. IEEE Transaction Visual Computing Graphics. 2002, 8(4): 373-382

[Pan2001] 潘荣江. Delaunay 三角剖分在随机聚合网屏生成中的应用. 山东大学硕士学位论文，2001

[Preparata1991] Franco P. Preparata and Michael Ian Shamos. Computational Geometry: An Introduction (3rd edition). New York : Springer-Verlag. 1991

[Press1992] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. Numerical Recipes in C: The Art of Scientific Computing (2nd edition). Cambridge University Press, New York, NY, USA, 1992

[Qi2007] Meng Qi, Chenglei Yang, Changhe Tu, Xiangxu Meng, Yuqing Sun. A GPU-Based Algorithm for Building Stochastic Clustered-dot Screens. In: LNCS 4841,Part I, pp.98-105, 2007(3rd International Symposium on Visual Computing (ISVC 2007), Lake Tahoe, NV, USA, November 26-28, 2007)

[Rong2006] Guodong Rong and Tiow-Seng Tan. Jump Flooding in GPU with

Applications to Voronoi Diagram and Distance Transform. In: Proceeding of symposium on Interactive 3D graphics and games. 2006, pp.109-116

[Rong2011] Guodong Rong, Yang Liu, Wenping Wang, Xiaotian Yin, Xianfeng David Gu, Xiaohu Guo. GPU-Assisted Computation of Centroidal Voronoi Tessellation. IEEE Transactions on Visualization and Computer Graphics. 2011, 17(3): 345-356

[Sack2000] J. R. Sack and J. Urrutia. Handbook of Computational Geometry. North-Holland Publishing Co., Amsterdam, The Netherlands, The Netherlands, 2000

[Sugihara2000] K. Sugihara, M. Iri, H. Inagaki, T. Imai. Topology-oriented implementation—an approach to robust geometric algorithms. Algorithmica. 2000, 27(1):5-20

[Sutherland1974] Ivan E. Sutherland and Gary W. Hodgman. 1974. Reentrant polygon clipping. Communications of the ACM. 1974, 171(1): 32-42

[Thiessen1911] Alfred H. Thiessen. Precipitation averages for large areas. Monthly Weather Review. 1911, 39: 1082- 1084

[Toth2001] G Fejes Tóth. A Stability Criterion to the Moment Theorem. Studia Scientiarum Mathematicarum Hungarica. 2001, 34: 209-224

[Tu2000] 屠长河，潘荣江，孟祥旭. 一种随机聚合网屏的生成算法. 计算机学报. 2000，23(9): 938-942

[Voronoi1908] Georges Voronoi. Nouvelles applications des paramètres continus à la théorie des formes quadratiques. deuxième memoire. recherche sur les parallelloèdres primitifs. Journal für die Reine und Angewandte Mathematik. 1908,134:198-287

[WangJY 2011] 汪嘉业，王文平，屠长河，杨承磊. 计算几何及应用. 北京：科学出版社。2011

[WangL2006] Lu Wang, Chenglei Yang, Meng Qi, Xiangxu Meng, Xiaoting Wang. Design of a Walkthrough System for Virtual Museum Based on Voronoi Diagram. In: Proceeding of 3rd International Symposium on Voronoi Diagrams in Science and Engineering (ISVD). 2006, pp.258-263

[WangXT2011] Xiaoting Wang, Chenglei Yang, Jiaye Wang, Xiangxu Meng. Hierarchical

Voronoi diagram-based path planning among polygonal obstacles for 3D virtual worlds. In: Proceeding of IEEE International Symposium on Virtual Reality Innovations (ISVRI 2011). 2011, pp.175-181

[YangCL2000] 杨承磊，孟祥旭，李学庆，龚斌，屠长河. 基于无向图的图像整体骨架表示模型及其算法. 计算机学报. 2000(3): 293-299

[YangCL2005] 杨承磊，汪嘉业，孟祥旭. 多边形的外部 Voronoi 图顶点和边数的上界. 计算机辅助设计与图形学学报. 2005, 17(4): 689-693

[YangCL2006a] Chenglei Yang, Pu Shi, Wei Zao, Lu Wang, Xiangxu Meng, Jiaye Wang. New Intersection Algorithm of Convex Polygons Based on Voronoi Diagrams. In: Proceeding of 3rd International Symposium on Voronoi Diagrams in Science and Engineering (ISVD). 2006, pp.224-229

[YangCL2006b] Chenglei Yang, Meng Qi, Xiangxu Meng, Xueqing Li, Jiaye Wang. A New Fast Algorithm for Computing the Distance between Two Disjoint Convex Polygons Based on Voronoi Diagram. Journal of Zhejiang University SCIENCE A. 2006, 7(9):1522-1529

[Yang2006CLc] 杨承磊，汪嘉业，孟祥旭. Upper Bounds on the Size of Inner Voronoi Diagrams of Multiply Connected Polygons. 软件学报. 2006, 17(7): 1527-1534

[YangCL2007] Chenglei Yang, Meng Qi, Jiaye Wang, Xiaoting Wang, Xiangxu Meng. Shortest Path Queries in a Simple Polygon for 3D Virtual Museum. In: LNCS 4705, 2007, pp.110-121 (7th Annual International Workshop on Computational Geometry and Applications (CGA'07), Kuala Lumpur, Malaysia, August 26-29, 2007)

[YangYJ2002] 杨义军，孟祥旭，杨承磊，汪嘉业. 一种基于 Delaunay 三角化的手写体文字细化方法. 中国图形图像学报. 2002, 7A(9): 938-944

[YangQ2005] 杨钦. 限定 Delaunay 三角网格剖分技术. 北京：电子工业出版社，2005

[Yap1987] Chee K. Yap. An O(nlogn) algorithm for the Voronoi diagram of a set of simple curve segments. Discrete and Computational Geometry. 1987, 2:365-393

[Yuan2011] Zhan Yuan, Guodong Rong, Xiaohu Guo, Wenping Wang. Generalized

Voronoi Diagram Computation on GPU. In: Proceeding of 8th International Symposium on Voronoi Diagrams in Science and Engineering (ISVD). 2011, pp.75-82

[Zeng2004] Zeng, Wei, Yang, Chenglei; Meng, Xiangxu; Yang, Yijun; Yang, Xiukun, Fast Algorithms of Constrained Delaunay Triangulation and Skeletonization for Band-images, In : Proceedings of SPIE The International Society for Optical Engineering, v 5403, n PART 1, Sensors, and Command, Control, Communications, and Intelligence (C31) Technologies for Homeland Security and Homeland Defense III. 2004, pp. 337-348

[Zeng2005a] 曾薇，孟祥旭，杨承磊，杨义军. 平面多边形域的快速约束 Delaunay 三角化. 计算机辅助设计与图形学学报. 2005. 17(9) : 1933-1940

[Zeng2005b] 曾薇. 多边形域的 Delaunay 三角化及其应用研究. 山东大学硕士学位论文. 2005

[Zeng2006] Wei Zeng, XiangXu Meng, ChengLei Yang, Lei Huang. Feature Extraction for Online Handwritten Characters Using Delaunay Triangulation. Computers&Graphics. 2006, 30(5): 779-786

[Zhao2013] 赵海森，杨承磊，吕琳，王筱婷，杨义军，孟祥旭. 多边形中的点可见性快速算法.计算机辅助设计与图形学学报. 2013, 25(3): 331-340

[Zhou2008] 周培德. 计算几何——算法设计与分析（第三版）. 北京：清华大学出版社，2008